VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Vyhledávání založené na lokální dostupnosti produktů v produktových katalozích

Search based on local product availability in product catalogs

Martin Vančura

Diplomová práce

Vedoucí práce: Ing. Radoslav Fasuga, Ph.D.

Ostrava, 2021

Abstrakt

Práce se věnuje problematice vyhledávání nad rozsáhlými kolekcemi dat, s využitím geografických poloh entit. V teoretické části textu je představena problematika zeměpisných souřadných systémů včetně reprezentace geografických lokací. Jako zdroj dat vhodný pro navrhované řešení je popsán český registr adresních míst. Dále je představen koncept prostorových dat a způsob jejich reprezentace v databázových systémech (relačních i jiných). Jsou zmíněny některé mapové a trasovací aplikace.

V druhé části je popsána demonstrační aplikace implementující navrhované řešení. Téma je představeno na vyhledávání produktů v e-commerce systému, kde jsou polohové atributy, spolu s ostatními vlastnostmi produktů, dostupné pro parametrizaci vyhledávání. Na tomto praktickém příkladu jsou ověřeny poznatky z teoretické části práce: zvolené databázové schéma, způsob sestavení dotazu s využitím informací o poloze prvků a integrace rozhraní mapové a trasovací aplikace.

Klíčová slova

prostorová data, prostorová databáze, zeměpisná poloha, zeměpisný souřadnicový systém, zeměpisná šířka, zeměpisná délka, hledání dle vzdušné vzdálenosti, hledání trasy, GPS

Abstract

Thesis deals with searching in large data collections using the geographical location of entities. The topics of geographic coordinate systems together with the representation of geographical locations are presented in the theoretical part of the thesis. Czech registry of addresses is described as a data source suitable for proposed solution. The overall spatial data concept and its means of representation in database systems, relational and others, is presented. Some of the existing map and routing applications are also mentioned.

The second part of the text describes a demonstrative application which implements the principles of the proposed solution. The topic is presented on the case of product search in e-commerce system where the location attributes are available for the query parameterization together with other product attributes. The purpose of this practical solution is to verify the findings from the theoretical part of the thesis: the chosen database schema, the way how the query is built by using the entities' location information and integration of map and routing application interfaces.

Keywords

spatial data, spatial database, geographical location, geographic coordinate system, latitude, longitude, direct distance search, route planning, GPS

Poděkování

Rád bych poděkoval vedoucímu diplomové práce, panu Ing. Radoslavovi Fasugovi Ph.D., za inspirativní a otevřené vedení příprav této práce, která by bez jeho podpory nevznikla. Dále bych chtěl poděkovat mé manželce za její podporu a trpělivost během celého mého studia.

Obsah

Seznam použitých zkratek a symbolů

AJAX	–	Asynchronous JavaScript and XML
API	–	Application Programming Interface
BSON	–	Binary JSON
CSV	–	Comma-separated Values
DOM	–	Document Object Model
ECEF	–	Earth-centered, Earth-fixed
EPSG	–	European Petroleum Survey Group
ETRS	–	European Terrestrial Reference System
GML	–	Geography Markup Language
GPS	–	Global Positioning System
HTTP	–	Hypertext Transfer Protocol
INSPIRE	–	Infrastructure for Spatial Information in the European Community
ISZR	–	Informační systém základních registrů
JDBC	–	Java Database Connectivity
JSON	–	JavaScript Object Notation
MBR	–	Minimum Bounding Rectange
OGC	–	Open Geospatial Consortium
OSRM	–	Open Source Routing Machine
REST	–	Representational State Transfer
RÚIAN	–	Registr Územní Identifikace, Adres a Nemovitostí
S-JTSK	–	Systém jednotné trigonometrické sítě katastrální
SQL	–	Structured Query Language
SRID	–	Spatial Reference System Identifier
SŘBD	–	Systém řízení báze dat
URL	–	Uniform Resource Locator
VDP	–	Veřejný Dálkový Přístup
VFR	–	Výměnný Formát RÚIAN
WGS	–	World Geodetic System

WKB	–	Well-Known Binary
WKT	–	Well-Known Text
XML	–	Extensible Markup Language

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Cílem práce je analyzovat možnosti a postupy pro implementaci vyhledávání, které bude kromě textových a numerických parametrů také brát v potaz geolokační informace hledaných entit. Konkrétním příkladem užití takového vyhledávání může být e-shopová aplikace poskytující hledání produktů dle dostupnosti na některém z výdejních míst. Uživatel může parametrizovat hledání takovým způsobem, kdy jeho vzdálenost od místa, kde se produkt nachází, bude omezena nějakou maximální hodnotou.

Teoretická část práce popíše základní principy zeměpisných souřadnicových systémů, ke kterým se budeme vracet i v dalších kapitolách textu. Dále bude představen vhodný zdroj lokačních dat. Největší prostor bude věnován popisu podpory prostorových dat v databázových systémech. Budou představeny obecné principy, jako jsou typy prostorových objektů, prostorové indexové struktury a operace nad prostorovými objekty. Budou popsány některé populární databázové systémy z hlediska úrovně podpory prostorových funkcí. V poslední části teoretického textu se budeme věnovat některým dalším aplikacím nabízejícím mapové a trasovací funkce.

V implementační části práce budeme demonstrovat zadání na příkladu e-commerce portálu Gloffer, který se věnuje nabídce nových a ojetých vozů na území České republiky. Pro trh s automobily je důležité evidovat a pracovat s lokačními údaji produktů, zde konkrétně s geografickými lokacemi poboček autobazarů a autosalónů, jelikož právě vzdálenost produktu od zákazníka může být jedním z důležitých kritérií pro uživatelské vyhledávání.

V rámci práce byla vyvinuta aplikace, pomocí které lze demonstrovat vyhledávání nad geolokačními údaji. V implementační kapitole je představen koncept demonstrační aplikace a jednotlivé oblasti řešení: vytvoření číselníků adres a geografických souřadnic, implementace podpůrných prvků pro přípravu vyhledávání a konečně vyhledávání s využitím geolokačních dat samotné.

Během vypracovávání této práce došlo k upřesnění původního zadání. Bylo záhodno využít uložených lokačních dat i pro jiné účely než pro vyhledávání produktů. Proto se práce ve svém závěru zaměřuje i na problematiku hledání nejkratších tras, a to za účelem prezentace navržené

okružní cesty od zákazníka k nalezeným lokacím a také pro zohlednění vzdálenosti silniční dráhy mezi vyhledávanými polohami.

1.1 Struktura dokumentu

Počáteční, vzhledem k číslování však druhá, kapitola textu se zabývá problematikou zeměpisných souřadnic, určování lokace v prostoru a získávání geografických lokací platných adres na území České republiky.

Třetí kapitola se zaměřuje na přístupy pro ukládání, manipulaci, analýzu a indexování prostorových dat v databázích. Jsou zde představeny některé databázové systémy z hlediska podpory funkcí pro práci s prostorovými daty.

Čtvrtá kapitola popisuje aplikace pro vizualizaci dat na mapě a pro získání trasovacích informací. Tato funkce je využita pro zpřesnění výsledků vrácených z databázového vyhledávání. Databáze pracuje se vzdušnou vzdáleností, ale z praktického hlediska je vhodnější pracovat s délkami trasy, tedy cesty vozidlem po silnicích. Aplikace třetích stran je také použita pro navržení trasy pro navštívení nalezených lokací a je zde stručně popsána varianta algoritmu řešícího „Problém obchodního cestujícího“ pro nalezení nejkratší možné okružní trasy.

Pátou kapitolou začíná popis implementační části práce, a to návrhem implementace. V úvodu kapitoly je cíl práce aplikován na praktický příklad specifického e-commerce systému a jsou popsány také jeho případy užití. Kapitola popisuje celkový koncept řešení včetně jednotlivých oblastí. Je navržena architektura demonstrační aplikace a databázové schéma.

Závěrečná šestá kapitola se již věnuje praktickým poznatkům z implementace samotné. Kapitola je členěna dle oblastí řešení a v popisu implementace jsou zmíněny hlavně body, které jsou specifické vzhledem k zadání práce.

V části příloh se nachází také kapitola představující příručku (ve formě krokového návodu) popisující instalaci, přípravu a spuštění aplikace.

Kapitola 2

Souřadnicové systémy a zdroje souřadnic

Cílem této kapitoly je představit základní principy zeměpisných souřadnicových systémů, jelikož následné kroky představené v této práci z těchto poznatků vychází. Principy a pravidla vycházející ze souřadnicových systémů mají vliv jak na výběr zdrojů polohových dat, tak na způsoby implementace ve všech dotčených informačních, aplikačních a databázových systémech, které pracují s daty reprezentující skutečná místa na zemi.

2.1 Souřadnicové systémy

Souřadnicový systém umožňuje popsat polohu bodu pomocí uspořádané n-tice hodnot, zvané souřadnice, v n-rozměrném prostoru. Přibližme si souřadnicové systémy používané v prostorových databázích a geografických informačních systémech [1].

2.1.1 Souřadnicové systémy zeměpisných souřadnic

Lokace je zadána pomocí zeměpisné šířky, zeměpisné délky (obě udávané ve stupních) a nadmořské výšky (udávané v metrech nad mořem).

Zeměpisná šířka udává úhlovou vzdálenost od rovníku. Pohybuje se mezi 0° (rovník) a 90° (zemské póly) a rozděluje se na severní a jižní dle příslušnosti k části polokoule. Zeměpisná délka udává úhlovou vzdálenost od základního poledníku. Pohybuje se mezi 0° (nultý poledník) a 180° (protilehlý poledník) a rozděluje se na východní a západní dle příslušnosti k části polokoule. Nadmořskou výškou se myslí svislá vzdálenost určitého místa na zemi ke střední hladině moře (teoreticky stanovena jako nulová výška nad zemským povrchem). Bývá reprezentována v topografických mapách, či v podobě rastrových dat. Pro určování polohy není informace o nadmořské výšce vyžadována.

Systém počítá s trojrozměrným tvarem Země. Pracuje se buď s tělesem typu koule či elipsoidu. Elipsoid více odpovídá skutečnému tvaru planety, jeho použití je tedy přesnější než v případě použití koule, ale z důvodu nerovnoměrného zakřivení zemského povrchu dochází i u něj k určité ztrátě přesnosti. Použití elipsoidu v analytických operacích nad souřadnicemi vede sice k přesnějším

výsledkům, nicméně sférické výpočty jsou jednodušší, a proto i rychlejší. Záleží tedy na konkrétním případu užití, zda má větší váhu přesnost či rychlost zpracování.

Referenční datum je množina hodnot definující polohu elipsoidu vzhledem k centru zeměkoule. Jde o referenční rámec definující počátek a orientaci os zeměpisné šířky a délky. Globální datumy lze použít pro celosvětové souřadnice, ale v některých případech může být vhodnější použít lokální datumy s větší přesnosti pro omezenou plochu země. Datum je nutné pro projekci sférického systému do roviny při vytvoření kartografických zobrazení [2].

2.1.2 Souřadnicové systémy rovinných souřadnic

Rovinné souřadnice určují polohu na mapě, což je výsledek převodu trojrozměrného povrchu Země do roviny. Při převodu se používá některý z geometrických objektů, které jsou do roviny rozvinutelné. Při převodu z kulovitého či elipsoidového tvaru však dojde k nějakému zkreslení se ztrátou přesnosti převáděných informací.

Součástí realizace zobrazení je i přepočet geografických sférických souřadnic pomocí definovaného datumu do dvourozměrného kartézského systému souřadnic. Směr na ose x od počátečního bodu (0, 0) je obvykle východní (pro kladné hodnoty x) či západní (pro hodnoty záporné). Analogicky pro osu y je směr severní a jižní.

Existuje více typů mapových zobrazení a záleží na konkrétním případu užití, jaké zobrazení vybrat. Jedním z kritérií může být míra zkreslení pozorované plochy, nebo kompatibilita zobrazení v uvažovaném systému.

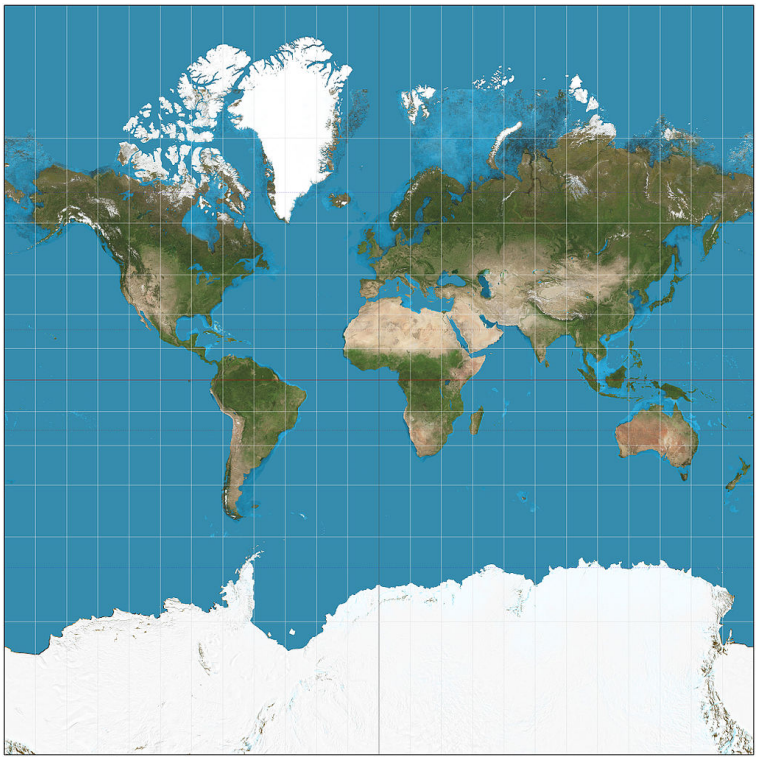
Webové mapové aplikace Google Maps, OpenStreetMap a Mapy.cz používají zobrazení Mercator, resp. jeho variantu zvanou „WGS 84/Pseudo-Mercator“ [3] [4]. Nevýhodou tohoto zobrazení je velké zkreslení polárních oblastí (rozloha Grónska na mapě zhruba odpovídá rozloze Afriky), viz obrázek 2.1.

2.1.3 Další možnosti reprezentace zeměpisných údajů

Existují i jiné způsoby pro určení poloh na zemi. Z jiných souřadných systému jmenujme například geocentrický kartézský souřadnicový systém ECEF s počátkem v hmotnostním těžišti Země.

V textu se také zmíníme formát Geohash, který patří do rodiny geodetických kódů. Dalším příkladem takového kódu je systém What3words, který rozděluje povrch země do sítě čtverců o délce stran 3 metry. Každý čtverec je pojmenován unikátní kombinací tří slov představujících adresu čtvercem vytyčené oblasti. Výhody použití slov pro adresaci jsou: snížení rizika špatného zadání adresy v případě manuálního zadání a snazší zapamatovatelnost. Pro někoho může být snadnější manuálně zapsat a zapamatovat si 3 reálná slova, než dvojici kódů (např. zeměpisné šířky a délky).

I údaje jako poštovní směrovací čísla či mezinárodní směrová (telefonní) čísla udávají zeměpisnou polohu v nějaké specifické míře detailů.



Obrázek 2.1: Zobrazení Mercator [5]

2.1.4 Souřadnicové referenční systémy

Souřadnicové referenční systémy jsou *„soubory matematických pravidel, která definují jednoznačné přiřazení souřadnic prostorovým informacím na zemském tělese, doplněný souborem parametrů, které definují polohu počátku, měřítko a orientaci souřadnicových os na zemském tělese.“* [6]

Například česká státní sféra pracuje s referenčními systémy S-JTSK (Systém jednotné trigonometrické sítě katastrální), ETRS (Evropský terestrický referenční systém), S-42/83 (Souřadnicový systém 1942) a WGS84 (Světový geodetický systém 1984) [7]. Referenční systém WGS84 je výchozím systémem technologie GPS [8].

Identifikátor souřadnicového referenčního systému je značen zkratkou SRID. Jedním z registrů těchto systémů je EPSG, ve kterém mají identifikátory referenčních systémů WGS84 a S-JTSK hodnoty 4326, resp. 5514.

V této práci se budeme zabývat hlavně systémy S-JTSK (primární zdroj číselníkových dat pracuje v tomto systému) a WGS84 (pro použití v mapových aplikacích a navigačních systémech pracující s GPS daty).

2.2 Zdroje souřadnic adres

Pokud mají být prostorové atributy zahrnuty do databázových dotazů, musí být součástí kolekce dat, nad kterou je vyhledávání prováděno. V opačném případě, pokud by implementace vyhledávání spoléhala na získání lokačních atributů teprve při provádění dotazu, například voláním API poskytovatele takových údajů, byla by doba vykonávání dotazu navýšena o čas potřebný pro vypořádání požadavku zmíněného API. Samozřejmě by také nešlo využít indexy nad lokačními údaji a optimalizace dotazu by obecně narážela na omezení ve formě úzkého hrdla pro komunikaci s externím rozhraním. Pokud se však údaje o polohách nachází v databázovém úložišti, nad kterým jsou konstruovány a spouštěny dotazy, zmíněné nedostatky mizí.

Cílem práce tedy bylo najít zdroj dat poskytující kompletní seznam platných poštovních adres na území ČR a jejich zeměpisných souřadnic. Správce aplikace by tak mohl vkládat záznamy o prodejcích pomocí zvolené platné adresy, u které je v databázi již uložen i údaj o lokaci. Zákazník by pak mohl při vyhledávání specifikovat výchozí polohu pomocí zadané adresy, bez znalosti konkrétních lokačních údajů.

Pokud mají být lokační údaje uložené ve vlastní databázi, pak musí toto vložení umožňovat i poskytovatel dat. Tato podmínka vychází z faktu, že různí poskytovatelé definují a omezují způsoby, jak může být s jejich daty nakládáno. Například Mapy.cz nabízí aplikační rozhraní pro získání GPS souřadnic pro zadanou adresu, ale ve svých podmínkách zakazuje tyto získané údaje komerčnímu subjektu volajícímu API rozhraní ukládat ve svých úložištích [9].

2.2.1 Registr územní identifikace, adres a nemovitostí (RÚIAN)

Jako zdroj dat, který splňuje stanovené podmínky, byl vybrán Registr územní identifikace, adres a nemovitostí (RÚIAN), *„jehož správcem je Český úřad zeměměřický a katastrální, vedeným Ministerstvem vnitra ČR, slouží k evidenci údajů o územních prvcích, údajů o územně evidenčních jednotkách, adresách, územní identifikaci a údajů o účelových územních prvcích. Vede referenční údaje o stavebních objektech, pozemcích, ulicích, katastrálních územích atd.“* [10] RÚIAN je jedním z centrálních registrů veřejné správy jejíž ostatní systémy se odkazují na adresy v něm vedené jako na referenční prvky [11] a vzhledem k tomuto postavení registru v ekosystému státní správy je zajištěna i validita poskytovaných dat.

K datům lze přistupovat pomocí aplikace Veřejný dálkový přístup (VDP) a získat je v XML formátu Výměnný formát RÚIAN (VFR). Data dostupná v systému VDP jsou aktualizovaná s měsíční periodicitou a jsou poskytovaná zdarma.

Po vyhledání požadovaných souborů VFR lze získat textový soubor s odkazy (URL) k umístění jednotlivých souborů. URL k souborům lze také získat pomocí webových služeb RÚIAN, které jsou poskytovány na eGON rozhraní Informačního systému základních registrů (ISZR). Automatizovaná aktualizace číselníkových dat není součástí řešení této práce, ale pro možné budoucí rozšíření funk-

cionality je doporučeno využít eGON rozhraní ISZR spolu s automatizovanou službou, která jednou měsíčně vyžádá z VDP aktuální data a po stažení provede analýzu dat, identifikaci rozdílů a jejich následnou aktualizaci v DB. Kompletní zdrojová data mají velké požadavky na diskový prostor (až 50 GB), proto je vhodné implementovat aktualizaci se získáváním pouze měsíčních rozdílových dávek, které jsou o několik řádů menší. VDP toto naštěstí umožňuje.

Další způsob získání dat z RÚIAN je pomocí rozhraní Stahovací služby ATOM, kde jsou některé sady číselníkových dat dostupné i ve formátu CSV. Jinou možnost představuje formát směrnice INSPIRE. Data adres sady INSPIRE poskytované ČÚZK, importované z RÚIAN, však k 15.2.2021 stále neobsahují 1,05%, t.j. 31083 adresních míst, které jsou dohledatelné v RÚIAN [12]. Data lze získat pomocí stahovacích služeb standardů WFS a WMS, které jsou podporované řadou GIS aplikací, či opět pomocí stahovací služby ATOM. I soubory dat INSPIRE jsou ve formátu v XML.

Jelikož data sady INSPIRE jsou neúplná, bylo by vhodné zvolit za zdroj adresních dat pro lokální číselník registr RÚIAN. U něj by bylo vhodné upřednostnit formát VFR před CSV, protože dává k dispozici kompletní data RÚIAN a ne jen jejich podmnožinu, jak je tomu u vybraných sad ve formátu CSV.

Datový model RÚIAN

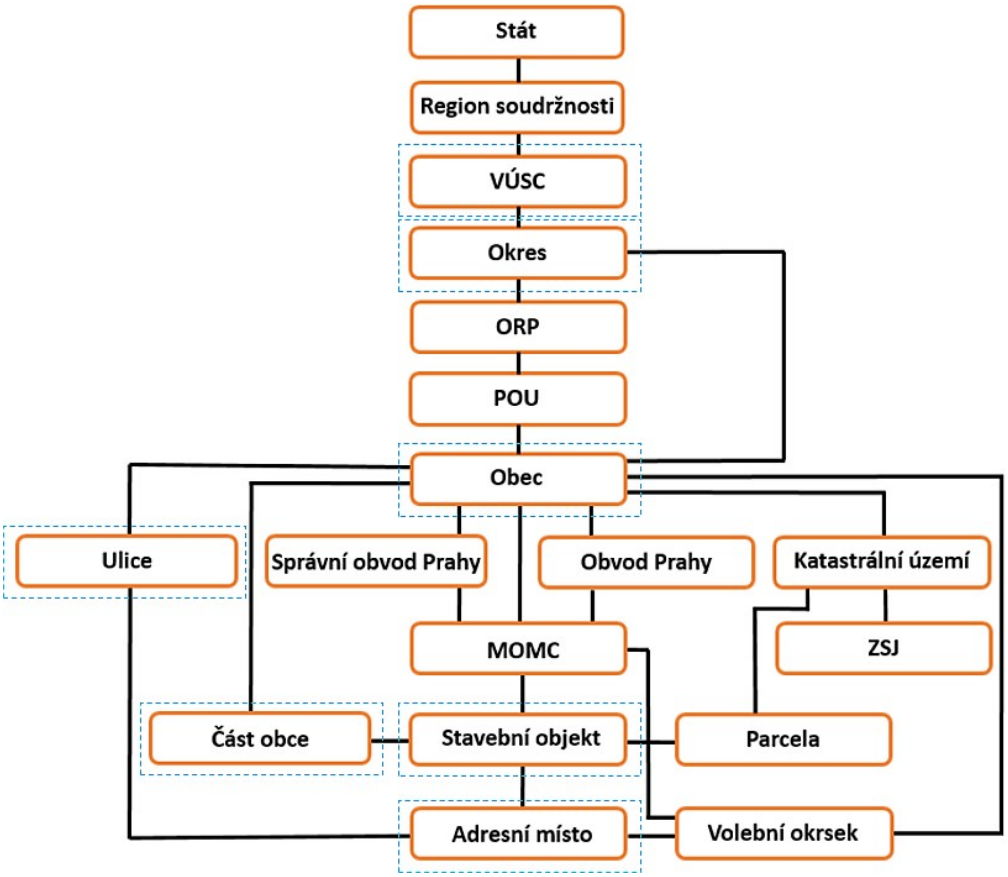
Číselník RÚIAN je komplexní a v rámci řešení zadání práce postačí zaměřit se jen na některé z nabízených atributů (např. atribut *připojení stavebního objektu k vodovodu* je pro účely zadání nepodstatný). Následující popis datového modelu RÚIAN se tedy omezuje jen na elementy, které jsou nutné pro řešený projekt.

Vazby mezi všemi entitami dostupnými v RÚIAN jsou viditelné na diagramu 2.2, kde jsou modrým rámečkem označeny prvky nutné pro řešení této práce. Diagram a popis dat odpovídá VRF verzi 3.1 z 1.1.2021 [13]. Atributy všech entity z RÚIAN lze rozdělit do těchto skupin:

- identifikátory záznamů. Zde spadají i atributy představující unikátní identifikátor jiné entity, pomocí kterých jsou mezi entitami realizovány vazby.
- atributy udávající konkrétní hodnotu či informaci objektu, který entita reprezentuje. Pro Adresní místo to může být např. číslo popisné, či název ulice pro entitu Ulice.
- atribut představující geometrické údaje. Atribut je ve formátu GML, který udává geometrické informace prvku, např. souřadnice objektu v případě Adresního místa v systému S-JTSK.
- technické atributy označující časovou platnost záznamu, jeho validitu, datum vzniku apod.

Pro získání kompletní adresy nějakého adresního místa je nutné pracovat s těmito entitami: Adresní místo, Stavební objekt, Část obce, Ulice, Obec.

- Adresní místo: Adresní místo je přiděleno ke Stavebnímu objektu a představuje jednoznačnou adresu místa. Obsahuje vazby na Ulici a Stavební objekt.



Obrázek 2.2: Datový model RÚIAN a zvýraznění použitých prvků [13]

- Stavební objekt: Stavebním objektem se rozumí budova nebo její části označené číslem popisným nebo evidenčním a budova bez čísla domovního (popisného/evidenčního). V případě, kdy adresní místo nemá přiřazenou ulici, přes kterou by šlo přiřadit adresní místo k obci, je nutné získat vazbu na atribut část obce, pomocí kterého lze pak identifikovat obec, do které spadá adresní místo. Právě Stavební objekt má registrovanou vazbu na Část obce.
- Ulice: obsahuje název ulice a vazbu na entitu Obec.
- Část obce: obsahuje název části obce a vazbu na entitu Obec.
- Obec: obsahuje název obce a vazbu na entitu Okres.

Postup sestavení kompletní poštovní adresy pro nějaké Adresní místo, je následující: v entitě Adresní místo jsou uloženy hodnoty číslo popisné, číslo orientační, písmeno čísla orientačního a PSČ a vazby na entity Ulice a Stavební objekt. V entitě Ulice získáme název ulice a vazbu na Obec. V entitě Obec získáme název obce a vazbu na Okres. V entitě Okres získáme název okresu. Z entity Stavební objekt se přes vazbu dostaneme k entitě Část obce. Ta obsahuje atribut název části obce.

Ostatní entity označené na diagramu poskytují informace, které pro sestavení poštovní adresy nejsou zapotřebí (např. Okres či VÚSC), ale pro případné budoucí použití je vhodné je v číselníku v databázi uchovávat.

2.2.2 Konverze souřadnicových systémů

RÚIAN používá souřadnicový systém S-JTSK, a proto je nutné je transformovat do systému WGS84 pro použití v mapových a navigačních aplikací pracujících s GPS souřadnicemi. Převodní algoritmy existují a střední polohová chyba transformace, ke které dochází vlivem zkreslení, je menší než 0,1 m [14]. Vzhledem k tomu, že projekt je zaměřen na lokalizaci adresních míst typicky představující nějakou budovu, chyby takovéhoto řádu jsou nepodstatné.

Kapitola 3

Prostorová data

Cílem kapitoly bude popsat jak obecné principy práce s prostorovými daty v databázích, tak i způsoby specifické pro některé z databázových systémů.

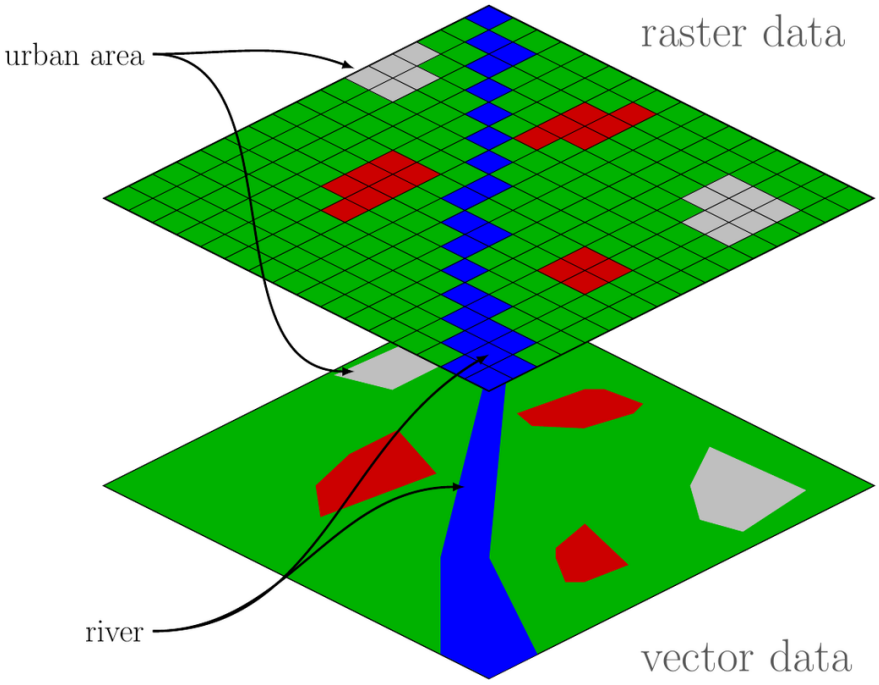
Prostorovými daty jsou *„data obsahující jak atributy, tak nějakou prostorovou informaci o umístění a tvaru objektu (tzv. geometrii).“* [15] Nemusí jít tedy pouze o data odpovídající reálným místům či oblastem ve světě, byť právě na tento typ dat se tato práce zaměřuje.

V kontextu systémů pro řízení báze dat, jsou prostorová data taková data, nad kterými lze volat prostorové funkce a vykonávat operace pomocí prostorových operátorů. Prostorovou funkcí může být například Contains, která vrací hodnotu 1, pokud se objekty předané v parametrech zcela překrývají, tedy pokud druhý objekt je zcela obsažen v prvním objektu. Prostorové funkce by měly zohledňovat typ prostorových dat a zmíněná funkce Contains pak musí brát v potaz, zda jde o bod, který je či není obsažen v druhém objektu, nebo zda jde o množinu bodů tvořících nějakou ohraničenou oblast. V takovém případě mohou všechny body prvního objektu ležet v oblasti dané druhým objektem, ale hranice, kterou body prvního objektu tvoří, může hranici druhého objektu protínat a pak funkce vrací hodnotu 0.

U některých SŘBD je možné prostorová data ukládat v prostorovém datovém typu. I prostorové funkce typicky pro své vstupní parametry požadují prostorové datové typy. Pro definici prostorových dat může být požadováno i určení souřadnicového systému, ke kterému se data vztahují. Uložení dat v prostorovém datovém typu může být také podmínkou pro indexaci dat.

Na datové typy prostorových dat lze nahlížet z mnoha pohledů. Rozlišujeme dva modely prostorových dat: vektorový a rastrový [16]. U obou typů dat lze ještě uvažovat kolekci atributů, která společně s vektorovými či rastrovými daty představuje kompletní prostorou informaci. O jaké atributy jde pak záleží na konkrétním určení prostorových dat. Příkladem takového atributu může být *velikost populace* pro nějakou oblast danou množinou bodů, které tvoří polygon.

V tomto projektu s rastrovými daty pracujeme pouze v případě zobrazení výsledků na mapě. Kromě následující kapitoly představující úvod do rastrových dat, se zbytek dokumentu zaměřuje pouze na práci s vektorovými prostorovými daty.



Obrázek 3.1: Porovnání rastrových a vektorových dat[17]

3.1 Rastrová data

Rastrová data jsou reprezentována jako množina buněk, které se vzájemně dotýkají a zcela tak vyplní nějaký sledovaný prostor. Prostorová informace je dána pozicí prvku vzhledem k ostatním prvkům. Buňky obsahují hodnoty, které prostor nějak popisují (např. výška terénu, hustota populace, naměřená teplota a další). Z rastru tak získáme celkový pohled na popisovaný fenomén, např. na místa s větší či menší hustotou obyvatelstva, či na srovnání teplot v různých místech.

Data popisovaná v jednotlivých buňkách rastru jsou buď diskrétní nebo spojitá, dle toho, zda jde jednoznačně určit hranici mezi sledovanými jevy. Diskrétními objekty mohou být například silnice či budovy. Spojitá je například výšková charakteristika krajiny, kdy je výsek území v rastrové tabulce reprezentován hodnotou odpovídající nadmořské výšce daného místa. Tuto výšku nelze pro všechny body vybrané buňky určit jednoznačně. Jiným příkladem spojitých dat je rastrová mapa koncentrace znečištění ovzduší, kde nelze přesně určit hranice, kde je koncentrace nějaké látky v ovzduší větší či menší a musí dojít k nějaké aproximaci.

Členění prostoru buněk může být buď pravidelné nebo nepravidelné. V prvním případě můžeme snadno každou buňku v mřížce popsat jednoznačnou adresou pozice (čísla řádku a sloupce v mřížce). V případě nepravidelných tvarů a velikostí buněk jsou jejich adresace a následné analýzy složitější.

Při využití rastrů v oblasti prostorových dat je nutné, aby data v mřížce byla organizována tak, aby bylo možné zobrazit rastr na reálných lokacích [18].

3.2 Vektorová data

Vektorová data mohou být oproti rastrovým datům přesnější, protože nejsou založena na mřížce a nejsou tedy limitována jejími rozměry. Skládají se z bodů, které mohou být spojeny hranami. V následujících sekcích této třetí kapitoly se budeme věnovat právě datům vektorového typu.

3.3 Specifikace OGC SFA

Organizace Open Geospatial Consortium (OGC) je sdružení více než 500 státních, neziskových a komerčních společností [19], které se věnuje také utváření standardů na poli geografických a geometrických dat. Jednou ze specifikací OGC je sada Simple Features Access (OGC SFA), jejíž cílem je definovat architekturu pro práci s prostorovými daty.

Implementační standard Simple Features Access, part 1: Common Architecture definuje obecnou architekturu geografických informací, datové typy – geometrické objekty, metody a operace [20].

Implementační standard Simple Feature Access, part 2: SQL Option definuje implementaci této obecné architektury pro jazyk SQL [21]. Jazyk SQL zde však není omezující podmínkou a principy standardu jsou implementované např. v MongoDB, což je představitel rodiny NoSQL databází. Ve specifikaci je navrženo použití prefixu ST_ pro pojmenování tříd a metod. Zkratka ST původně znamenala Spatial & Temporal [22], byť např. v PostGIS dokumentaci je odvozena jako Spatial Type [23]. Tato specifikace se v dokumentacích některých databázových systémů také označuje zkratkou SFS.

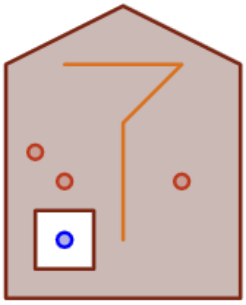
Všechny prostorové databázové systémy, které budou dále v textu popsány, implementují prostorové rozšíření také dle této specifikace. Proto i následující popis představených objektů a metod z ní bude vycházet.

3.4 Datové typy prostorových objektů

Kompletní seznam definovaných geometrických typů specifikace OGC SFA je zobrazen na třídním diagramu na obrázku 3.3. Věnujme se nyní jen těm základním typům, které jsou implementovány ve všech prostorových databázích, které budeme v tomto dokumentu srovnávat:

- Point
- LineString
- Polygon
- MultiPoint
- MultiLineString

- MultiPolygon
- Geometry
- GeometryCollection



Obrázek 3.2: Ukázka základních prostorových objektů Point, LineString, Polygon dle dotazu 3.2

Point (bod) je n-tice hodnot, tedy souřadnice, udávající polohu v prostoru. V případě reprezentace objektů reálného světa v prostorových datech mohou body představovat například budovy, adresní místa či centra měst. Bod nemá žádnou délku ani nezabírá žádnou plochu. Jeho dimenze je 0.

LineString (linie) je tvořena uspořádanou kolekcí souřadnic, které po spojení vytvoří křivku. V reálném světě může představovat ulici či říční tok. Ani uzavřená linie nedefinuje žádnou uzavřenou plochu. Její dimenze je 1.

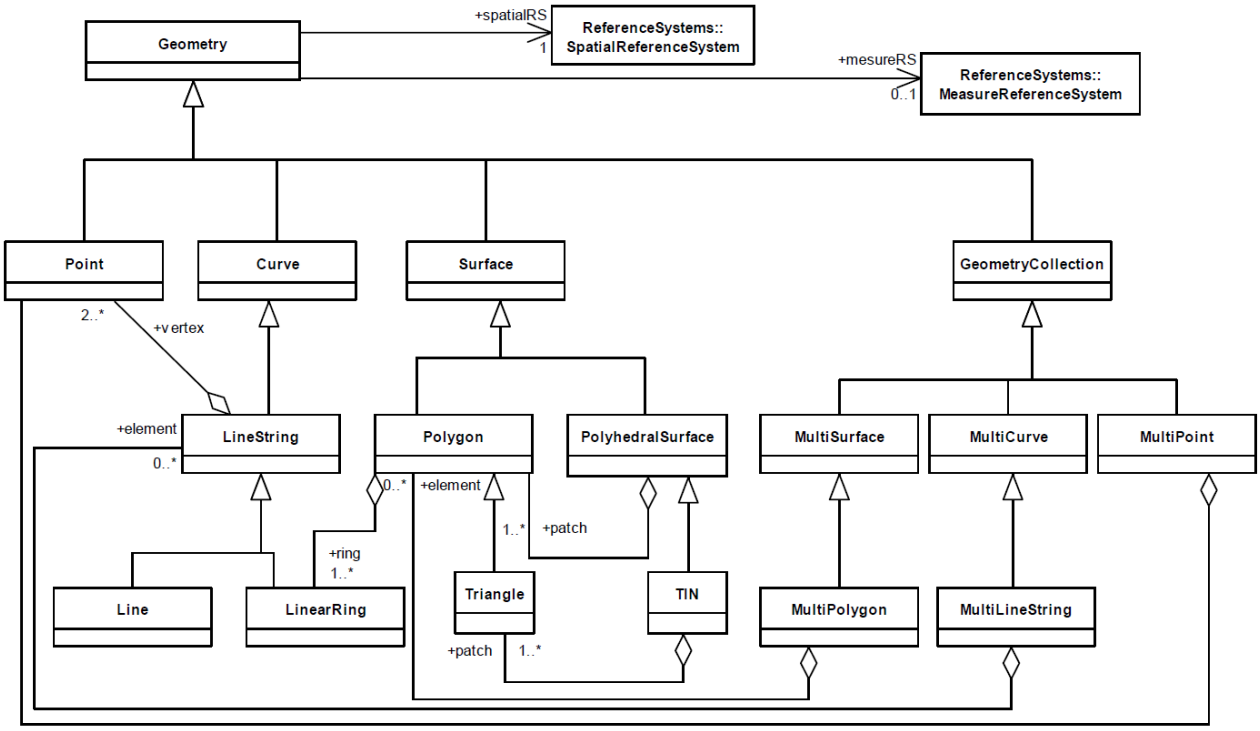
Polygon (plocha) je tvořen uspořádanou kolekcí souřadnic, u kterých se předpokládá, že poslední bod kolekce je totožný s prvním, čímž po spojení souřadnic vznikne topologicky uzavřený mnohoúhelník, reprezentující nějakou oblast. Takové oblasti odpovídají například městům či pozemkům. Polygon může obsahovat i jiný polygon. Výsledná plocha obsahuje všechny body obou polygonů, kromě těch, které se nacházely v obou mnohoúhelnících. V teorii množinových operaci jde o symetrickou diferenci. Dimenze polygonu je 2.

Dalšími odvozenými typy jsou MultiPoint, MultiLineString a MultiPolygon, které umožňují zadat data pomocí většího množství uvedených atomických objektů. Každý takový objekt však jako celek představuje jednu informaci. Například typem MultiPolygon může být definováno souostroví, které se sice skládá z jednotlivých ostrovů, polygonů, ale pouze všechny dohromady tvoří validní reprezentaci prvek reálného světa.

Datový typ Geometry je zobecněním ostatních tříd a jeho instance může být vytvořena jen pomocí definice některého z ostatních objektů a jejich kombinací. GeometryCollection je kolekcí typů Geometry, viz příklad 3.1.

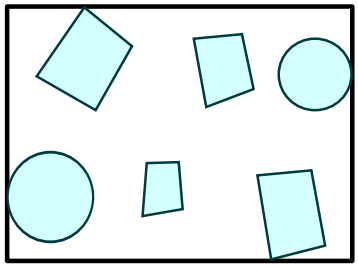
```
select ST_GeomFromText('GEOMETRYCOLLECTION(
                    POINT(-1 0),
                    LINESTRING(0 0, 0 2, 1 3, -1 3))');
```

Listing 3.1: Ukázka dotazu pro vytvoření objektu GeometryCollection, který se skládá z prostorových objektů Point a LineString.



Obrázek 3.3: Hierarchie třídy Geometry z OGC SFA [20]

3.4.1 Minimální ohraničující obdélník (MBR)



Obrázek 3.4: Ukázka MBR pro množinu polygonů [24]

Minimální ohraničující obdélník (angl. minimum bounding rectangle, MBR), též obálka (angl. envelope), je obdélník, který ohraničuje jeden a více prostorových objektů, včetně jiných MBR. Ohraničení je takové, že vymezená plocha je minimální a neexistuje jiný obdélník, který by ohraničoval totožné objekty a zároveň obsahoval menší plochu. Pro jeho definici postačí 2 dvojice souřadnic udávající jeho levý spodní a pravý horní vrchol [25].

0-rozměrný bod je totožný se svou obálkou. Pro obdélník obsahující množinu takových bodů platí, že všechny body z kolekce musí ležet v MBR nebo na jeho hranách. Pro 1-rozměrné linie a křivky a pro 2-rozměrné plochy (mnohoúhelníky) platí, že nejen všechny body, ze kterých se objekt skládá, patří do MBR, ale také hrany, které body spojují, zcela leží ve stejné obálce a žádná spojnice neprotíná hranici ohraničujícího obdélníku, ale může být součástí této hranice, viz obrázek 3.4. MBR objekty jsou využívané při indexaci prostorových dat, pro rychlé vyhledávání na základě vztahů mezi objekty a jejich obálkami, či pro rychlou vizualizaci hierarchie a poloh objektů.

Ilustrujme výhody použití MBR na příkladu vyhledávání bodů, které spadají do oblasti vytyčené nějakým polygonem. Naivní implementace algoritmu by pro každý bod zjistila, zda spadá do výchozí oblasti. V případě členitých hranic polygonu bude toto ověření výpočetně náročnější, než když bude tato oblast ve tvaru obdélníku, kde přítomnost bodu v této oblasti můžeme ihned ověřit porovnáním souřadnic bodu se souřadnicemi vrcholů obdélníku. S využitím MBR objektů, tedy pokud i body samotné jsou shlukovány do obálek, pak první úroveň filtrace vhodných kandidátů může rovnou vyloučit potenciálně velké množství bodů, jejichž MBR neprotínají MBR výchozího polygonu, protože v takovém případě ani body těchto ohraničujících obdélníků nemohou ležet uvnitř tohoto mnohoúhelníku.

3.5 Metody a relace prostorových objektů

Standard OGC SFA specifikuje i metody a operátory pro práci s prostorovými objekty [20]. Zmíníme základní metody, kterým se budeme v textu ještě věnovat a to hlavně při srovnání možností prostorových databází, a metody vhodné pro řešení zadání této práce. Pokud není uvedeno jinak, jedná se o metody, a tedy se předpokládá, že jsou volány nad nějakým objektem jehož atributy jsou v metodě dostupné. V dokumentaci OGC SFA je tento objekt označen klíčovým slovem *this*.

Vybrané základní metody:

- Envelope: vrací MBR objektu
- AsText: vrací definici objektu ve formátu WKT
- AsBinary: vrací definici objektu ve formátu WKB

Vybrané metody zjišťující relace mezi prostorovými objekty (vrací boolean hodnoty true a false):

- Equals: zjistí, zda je objekt prostorově shodný s jiným objektem

- Disjoint: zjistí, zda je objekt prostorově disjunktní s jiným objektem
- Intersects: zjistí, zda objekt prostorově protíná jiný objekt
- Touches: zjistí, zda se objekt prostorově dotýká jiného objektu. Dotykem se myslí situace, kdy se protínají hranice objektů, ale ne jejich vnitřní plochy.
- Crosses: zjistí, zda hranice objektu protínájí hranice jiného objekt
- Within: zjistí, zda je objekt uvnitř jiného objekt. Žádná z hran vnitřního objektu neprotíná hrany vnějšího objektu. Pokud se objekt skládá z bodů, pak všechny leží uvnitř oblasti vnějšího objektu a ne na jeho hranici.
- Contains: inverzní metoda k Within. Platí: $(a.Contains(b) = 1) \Leftrightarrow (b.Within(a) = 1)$
- Overlaps: zjistí, zda objekt částečně překrývá jiný objekt. Jiný objekt nesmí být zcela překryt, protože pak by šlo o relaci Contains. Hranice objektů se tedy protínají.

Vybrané metody pro prostorovou analýzu:

- Distance: vrací nejkratší vzdálenost mezi dvěma zadanými objekty. V případě objektů linie či plocha se bere v potaz nejkratší vzdálenost mezi libovolným místem na hranách polygonu, či přímo na linii, aniž by tento bod byl explicitní součástí definice objektu.
- Buffer: vrací nový prostorový objekt představující všechny body, které leží ve vzdálenosti shodné, nebo menší, než je vzdálenost zadaná parametrem metody.

3.6 Formáty reprezentace prostorových dat

Z existujících formátů pro reprezentaci prostorových dat představme ty, které jsou využívány v databázových systémech, které budou dále v textu srovnány.

3.6.1 WKT, WKB

Well-known text (WKT) je značkovací jazyk vytvořený pro popis definici a popis prostorových informací. Syntaxe se skládá z názvu typu objektu, za kterým následuje v závorkách množina souřadnic, případně kolekce dalších vnořených typů. Souřadnice jsou odděleny čárkou a jednotlivé složky souřadnic mezerou.

Pro ilustraci si ukažme MySQL dotaz 3.2 s funkcí `ST_GeomFromText`, do které WKT definice objektů vstupuje jako parametr.

```
select ST_GeomFromText('POINT(-1 0)'),
     ST_GeomFromText('MULTIPOINT((-1.5 1.5), (-1 1), (1 1))'),
     ST_GeomFromText('LINESTRING(0 0, 0 2, 1 3, -1 3)'),
```

```
ST_GeomFromText('POLYGON((-2 -1, 2 -1, 2 3, 0 4, -2 3, -2 -1),
                (1.5 -0.5, -0.5 -0.5, -0.5 0.5, -1.5 0.5, -1.5 -0.5))');
```

Listing 3.2: Ukázka dotazu pro vytvoření prostorových objektů Point, MultiPoint, LineString, Polygon. Výsledek je zobrazen na obrázku 3.2.

Well-known binary (WKB) je binární vyjádření informace formátu WKT. Takto serializovanou hodnotu lze využít například při implementaci automatizované komunikace mezi systémy. Hodnota se skládá ze sekvence bajtů, kde první bajt představuje endianitu, další 4 bajty pak o jaký prostorový datový typ jde a zbylé bajty samotné souřadnice objektu.

Oba formáty jsou specifikované v OGC SFA.

3.6.2 GeoJSON

Jde o rozšíření formátu JSON, pomocí kterého lze reprezentovat prostorové datové typy dle specifikace OGC SFA. Geografický souřadný referenční systém, pro souřadnice objektů zadaných pomocí GeoJSON, je WGS84 [26].

3.6.3 Geohash

GeoHash je kód reprezentující nějakou oblast či lokaci pomocí řetězce číslic a písmen. Čím je kód delší, tím přesnější je lokace, která reprezentuje zakódovaný objekt. Pro snížení nároků na uložení dat lze tedy kód od konce zkracovat, což s sebou nese zhoršenou přesnost kódu.

Pro zakódování určité lokace je celkový prostor rekurzivně dělen do mřížky. Každé buňce je přidělena hodnota délky 1, která je na dané úrovni mřížky unikátní. Na osmé úrovni pro lokace určené kódem o osmi znacích je maximální odchylka vůči skutečné lokaci přibližně 19 metrů [27].

Různé objekty, jejichž prefixy jejich kódů se shodují, budou sobě navzájem prostorově blízko. Míra blízkosti závisí na délce shodného prefixu. Nicméně tato úměra neplatí z opačné strany – tedy ne všechny blízké lokace musí mít shodné prefixy geohash kódů. Obě tato pravidla vyplývají ze způsobu, jakým se kódování provádí.

3.6.4 GML

GML je jazyk ve formátu XML definovaný organizací OGC [28]. Objekty se definují jako elementy (např. `<gml:Polygon>`), které obsahují vnořené elementy pro zadání souřadnic (`<gml:Pos>` nebo množin souřadnic (`<gml:coordinates>`, `<gml:posList>`)).

Na rozdíl od formátu GeoJSON, pro GML není určen jeden výchozí souřadný systém. Ten musí být definován atributem `srsName` pro element definující konkrétní objekt.

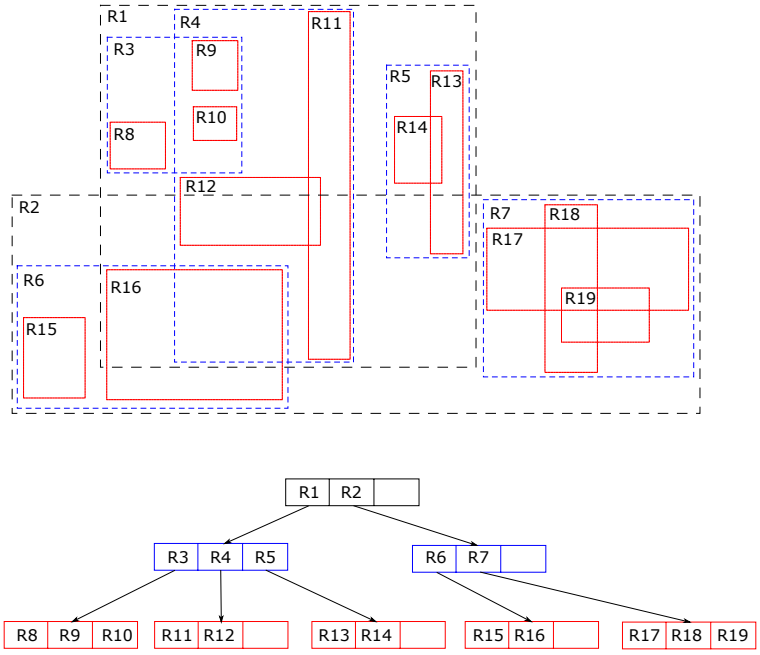
3.7 Indexace prostorových dat

Konvenční metody indexů pro ordinální datové typy nelze pro indexaci prostorových dat použít, dokud nejsou prostorová data nějak transformována na hodnoty, které lze uspořádat a uložit do stromové struktury (např. typu B-Tree). Jiným přístupem je vytvořit index tak, že se mezi evidovanými objekty zohlední jejich vzájemné vztahy a polohy vůči ohraničujícímu prostoru. Tento prostor se dále rekurzívně dělí na podoblasti. Pokud hledaný prostorový objekt nenáleží, neprotíná nebo se nedotýká oblasti na některé vyšší úrovni, pak není třeba prozkoumávat její podoblasti. Hledaný objekt se v nich nemůže nacházet.

Představíme si pouze indexové struktury, které jsou používány konkrétními databázovými systémy dále popsanými v tomto dokumentu.

3.7.1 R-Tree

Stromová struktura R-Tree [29], která je implementována v populárních relačních databázových systémech, jako jsou MySQL, PostGIS, ORACLE či Microsoft SQL Server.



Obrázek 3.5: Ukázka R-Tree struktury pro 2D obdélníky [30]

R-Tree je, podobně jako B-Tree, balancovaný strom. Je vhodný pro zrychlení přístupu k prostorovým a obecně k vícerozměrným datům. R-Tree využívá pro organizaci dat v uzlech stromu objekty MBR (odtud „R“ odpovídající anglickému výrazu rectangle). Konkrétní data jsou uložena v listech. Objekty k sobě blízké se shlukují do MBR objektů, které jsou uložené v nadřazeném uzlu

o jednu úroveň stromu výš. Tyto obdélníky jsou pak opět dle vzájemné vzdálenosti shlukovány do dalších MBR objektů.

Tato struktura zefektivní vyhledávání konkrétních dat následujícím způsobem: algoritmus postupuje od kořene stromu a porovná polohu hledaného objektu s hranicemi ohraničujících obdélníků v uzlech. Obdélníky, do kterých hledaný objekt nemůže spadat, jsou ignorovány a jejich potomci už nemusí být procházeny.

I ohraničující obdélníky na stejné úrovni se mohou překrývat. Může se tedy stát, že při procházení potomků nějakého obdélníku nebude hledaný objekt nalezen a hledání musí pokračovat průchodem dalším obdélníkem na úrovni toho právě zpracovaného.

V případě nejnepříznivějšího uložení dat nemůže R-Tree zaručit dobrý výkon a v nejhorším případě může být složitost vyhledávání $O(n)$. Průměrná složitost vyhledávání, pokud se obdélníky příliš nepřekrývají, je $O(\log_M n)$, kde M je maximální počet prvků v uzlu [31].

Pro vložení dat se strom prochází od kořene a hledá se takový ohraničující obdélník, jehož objekty v něm obsažené jsou vkládanému objektu nejblíže a MBR bude tedy nutné rozšířit méně, než jiné, pokud vůbec. Rozšíření obdélníků není žádoucí, protože může vést k většímu vzájemnému překryvu obdélníků, což má negativní dopad na složitost vyhledávání. Pokud je nutné rozdělit uzel, používají se pro nalezení ideálního dělení heuristické metody.

Tato metoda vytváření indexu se označuje jako „data-driven“ [32], kde struktura indexu závisí na podobě samotných dat, například oproti indexu Grid (viz 3.7.3).

3.7.2 k-d Tree

Struktura k-d Tree je určena pro indexaci vícerozměrných dat a lze ji tedy využít i pro prostorová 2-d data. Jde o strom, při jehož vytvoření se k-rozměrný prostor rozdělí pomocí nadroviny na maximálně dvě části (jde tedy o binární strom). V případě dvourozměrného prostoru bude nadrovinou úsečka rovnoběžná s jednou z os plochy. Rozdělené části tvoří uzly stromu a rekurzivně se dělí dále. Konkrétní body indexovaného prostoru jsou uloženy v listech.

Prostor se dělí tak, že se zvolí jeden z atributů dat, tedy jedna z k dimenzí, a dle mediánu těchto hodnot všech bodů v aktuální větvi stromu se prostor rozdělí. Všechny prvky se přesunou do levé či pravé větve dle toho, zda jejich složky, dle kterých se prostor rozdělil, jsou menší nebo větší než medián. Na následující úrovni se zvolí jiná dimenze prvků, jejíž medián určí další dělení prostoru.

Vyhledávání bodů funguje na podobném principu, jako u již zmíněného R-Tree a u stromových struktur obecně: postupujeme od kořene stromu a dle příslušnosti lokace hledaného bodu k jedné z částí uzlu postupujeme hlouběji až k uzlu, jehož potomci jsou listy, které musí být prohledány pro nalezení hledaného bodu. Prostory dělené v uzlech jsou na rozdíl od R-Tree disjunktní. Průměrná složitost vyhledávání je $O(\log n)$, v nejhorším případě $O(n)$ [33].

k-d-b Tree

Jak může být patrné z názvu, jde o stromovou strukturu vycházející z k-d Tree s tím rozdílem, že jde o balancovaný strom. Podobně jako u B-stromu, uzly jsou uloženy jako stránky. Ty mohou obsahovat buď kolekci dvojic (oblast, potomek) nebo (bod, umístění). V prvním případě se „oblastí“ myslí rozdělená část prostoru a „potomkem“ je ukazatel na stránku patřící do uvedené oblasti. V druhém případě pak „bod“ představuje konkrétní indexovaný objekt a „umístění“ ukazatel na jeho fyzickou lokaci [34].

Bkd Tree

Struktura Bkd Tree se skládá z množiny balancovaných k-d stromů [35].

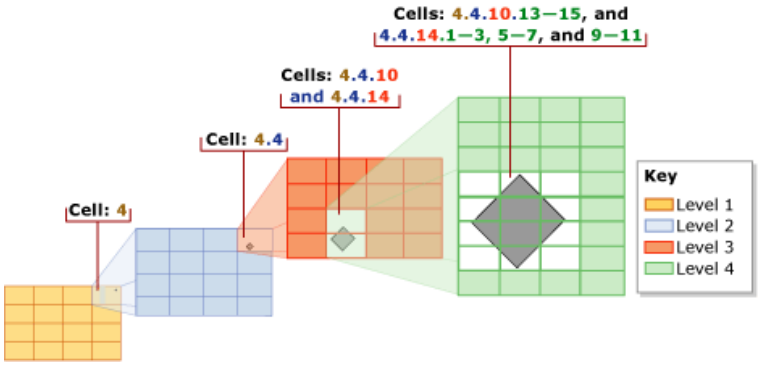
3.7.3 Grid

Při tvorbě indexové struktura Grid se prostor rozloží do mřížky, tvořené buňkami stejných rozměrů. Prostor definovaný každou buňkou je pak dále podobně rekurzivně rozčleněn. Počet úrovní není v zásadě omezen, záleží na konkrétní implementaci. Například Microsoft SQL Server (MSSQL) pracuje s čtyřmi úrovněmi, zatímco IBM Db2 maximálně se třemi. I počet buněk v každé úrovni může být pro různé implementace či instance indexu rozdílné. MSSQL umožňuje zvolit mezi variantami 4x4, 8x8, 16x16. Čísla buněk se ukládají do B-Tree indexu.

Při vytváření indexu nad nějakou tabulkou s prostorovými daty algoritmus pro první úroveň zjistí, které objekty se dotýkají buněk stejné úrovně. Pak se rekurzivně zjišťuje totéž pro všechny buňky, kromě těch, do jejichž oblastí žádný objekt nezasahoval.

Na rozdíl od struktury R-Tree jde o metodu „space-driven“ [32]. Nejdříve se vytvoří mřížka nehledě na podobě dat, které se budou indexovat. Struktura mřížky pak zůstává neměnná.

Index urychlí například hledání objektů, které se dotýkají. Pokud nesdílí žádnou z buněk na první úrovni, je jasné, že toto nemůže platit na nižších úrovních, které tedy není nutné procházet. Tím se vždy na každé úrovni odfiltrují oblasti, u kterých není nutné polohy dále analyzovat [36].



Obrázek 3.6: Ukázka Grid struktury s jedním objektem [36]

3.8 Podpora prostorových funkcí ve vybraných databázových systémech

Tato kapitola má za cíl popsat a srovnat podporu pro práci s vektorovými prostorovými daty v některých relačních a NoSQL databázích (dokumentových a vyhledávacích). Text si neklade za cíl nahradit uživatelské manuály těchto systémů, ale popsat klíčové aspekty implementace prostorových rozšíření. Z obsáhlé nabídky funkcí, které systémy nabízí, jsou popsány pouze ty, které by šly použít pro řešení této práce, nebo ty, které jsou pro popisovaný systém v něčem specifické.

Od prostorových databází očekáváme podporu datových typů vhodných pro uložení prostorových dat. Data musí být možné indexovat tak, aby indexové struktury umožnily efektivní vykonávání prostorových dotazů, jako například zjištění vzdálenost mezi zadanými objekty, nebo nalezení všech objektů v určité vzdálenosti od nějakého výchozího bodu. Prostorové databáze by měly poskytovat funkce a operátory pro zjištění vzájemných vztahů mezi objekty, tedy například, zda bod leží uvnitř polygonu, či zda se linie protínají. Vzhledem k cíli práce se budeme soustředit na datové typy a funkce použitelné pro práci s geografickými objekty a ne pouze s objekty zadané v euklidovském prostoru. Systémy by měly umožnit transformace mezi souřadnicovými systémy nebo alespoň podporovat systém WGS84.

Popisované systémy MySQL, MongoDB a Elasticsearch byly zvoleny z důvodu uvažovaného rozšíření aplikace Gloffer, která je na těchto systémech postavena, o poznatky obsažené v této práci. Zároveň jde o systémy, které jsou ve svých kategoriích (relační, dokumentová NoSQL a vyhledávací NoSQL databáze) stále populární[1]. U MySQL uvádíme ve zkratce ještě její open source nástupnickou větev MariaDB.

Dalším představeným systém je PostGIS, což je nadstavba relační databáze PostgreSQL. Systém byl zvolen, jelikož jde o open source software s licencí vhodnou i pro komerční účely, a protože je na něm postaveno několik populárních GIS aplikací, jako ArcGIS či OpenStreetMap [37].

V závěru kapitoly ještě krátce zmíníme Oracle, Microsoft SQL Server a IBM DB2 jako zástupce velkých hráčů na poli relačních databázi.

[1]Dle webu https://db-engines.com/en/ranking obsadily zmíněné systémy v žebříčku popularity databázových systémů prvních deset pozic (stav ke dni 1.4.2021).

1. Oracle
2. MySQL
3. Microsoft SQL Server
4. PostgreSQL
5. MongoDB
6. IBM Db2
7. Redis
8. Elasticsearch
9. SQLite
10. Microsoft Access

3.8.1 MySQL

Text této kapitoly čerpá z oficiální dokumentace MySQL [38].

MySQL od verze 5.7 implementuje rozšíření pro práci s prostorovými daty na základě specifikace OGC SFA. V této verzi však není možné pracovat v souřadných referenčních systémech. S prostorovými daty se manipuluje ve dvourozměrné kartézské soustavě, ale chybí zde podpora projekce z jiných souřadných systémů. Nelze tedy například správně počítat vzdálenosti mezi objekty reálného světa bez náležité projekce do roviny, kdy se vezme v potaz zakřivení Země, či kalkulace přímo na kouli nebo elipsoidu.

Podpora souřadnicových systému je v MySQL dostupná od verze 8.0. Jsou podporovány jak ty projektované do roviny, tak i geografické souřadnicové systémy. Ve verzi 8.0.22, na které byla vyvíjena demonstrační aplikace, je dostupných více než 5000 souřadných systému. MySQL navíc používá i systém SRID 0, který reprezentuje abstraktní rovinu bez daných jednotek. Při práci s prostorovými daty se tento systém v MySQL bere jako výchozí, a proto je nutné dbát na správnou a úplnou specifikaci těchto souřadných systému při definici a manipulaci s daty, které z konkrétních souřadných systémů pochází.

Kdykoliv je v dalších částech textu zmiňován systém MySQL, myslí se tím ve verzi 8.0 a novější, pokud není uvedeno jinak.

Datové typy

MySQL podporuje 8 základních prostorových datových typů dle specifikace OGC SFA. MySQL prostorové informace obecně označuje v dokumentaci jako geometrie (Geometry), což je ve shodě se zmíněnou specifikací, kde třída typu Geometry je v hierarchické struktuře na prvním místě a ostatní třídy jsou z ní zděděné. Instance typu Geometry lze vytvořit jen instancí jiného prostorového typu, viz 3.1. Pro zadání a výpis prostorových datových typů MySQL podporuje formáty WKT a WKT.

Indexy

Prostorový index je v MySQL implementován jako R-Tree, není však dostupný ve všech typech úložišť.

Prostorové datový typy a operace jsou dostupné pro uložiště MyISAM, InnoDB, NDB a ARCHIVE. Poslední dvě jmenované však nepodporují prostorové indexy a nejsou tedy vhodné pro dotazy, u kterých je nutné sledovat a optimalizovat dobu vykonání. Výjimkou mohou být jen dotazy s konkrétní hledanou hodnotou souřadnice v predikátu, kde může být použit B-tree index. MyISAM podporuje uložení dat pouze v souřadných systémech projektovaných do roviny, zatímco InnoDB podporuje navíc i geografické souřadné systémy. InnoDB je tedy nejvhodnější úložiště pro práci s prostorovými daty.

Vybrané funkce

Většina prostorových funkcí začíná prefixem `ST_`, tak jako v mnoha jiných DB systémů s rozšířením pro prostorová data. Názvy funkcí pracujících s MBR objekty začínají prefixem `MBR`.

Při volání funkcí analyzující dvě a více geometrií platí, že všechny tyto prostorové objekty musí být definované ve shodném souřadném systému. Je-li výstupem takové funkce opět geometrie, tak ta je automaticky definována v totožném souřadném systému.

Zaměřme se nyní na funkce pro podporu prostorové analýzy:

- `ST_Buffer(g, d)`: využití funkce `ST_Buffer` se nabízí pro vyhledávání všech bodů ve vzdálenosti d od bodu g. Pomocí funkce `ST_Within`, která určuje, zda geometrie leží uvnitř této kruhové obálky, lze získat hledané objekty. Bohužel funkce `ST_Buffer` v MySQL 8.0.23 stále podporuje pouze souřadný systém SRID = 0 a nelze tak být použita pro prostorové objekty reálného světa. Pokud se `ST_Buffer` volá nad daty definovanými v jiných souřadných systémech, funkce vrátí chybu `ER_NOT_IMPLEMENTED_FOR_GEOGRAPHIC_SRS`.

- `MBRContains(g1, g2)`: funkce zjistí, zda MBR pro geometrii g1 zcela obsahuje MBR pro geometrii g2. Funkce podporuje geometrie zadané v geografickém souřadném systému a také dokáže použít prostorový index. Může být tedy použita pro filtraci dat, kdy obálkou objektu g1 a pomocí indexu omezíme prohledávaný prostor.

- `ST_Distance_Sphere(g1, g2)`: funkce vrací nejkratší vzdálenost (v metrech) mezi zadanými objekty. Přípustné kombinace tříd objektů předaných v parametrech jsou (Point, Point) nebo (Point, MultiPoint). Na rozdíl od `ST_Distance`, tato funkce počítá vzdálenost na povrchu koule a je vhodná pro vstupy představující zeměpisné souřadnice.

Kombinace funkcí `MBRContains` a `ST_Distance_Sphere` je použita právě pro vyhledávání v implementované demonstrační aplikaci, viz ukázka dotazu v příloze D na stránce 98, a také pro dotaz s obálkou v porovnání výkonů MySQL a PostGIS systémů v kapitole 3.8.7.

Návrh schématu

Pro vytvoření sloupců s prostorovými datovými typy je nutné zvážit následující možnosti.

Konkrétní prostorový datový typ: jaký prostorový datový typ zvolit samozřejmě vychází z analýzy požadavků. Již bylo zmíněno, že typy Geometry a GeometryCollection jsou zobecněním jiných a lze je využít při definici datového typu sloupce. Jelikož prostorové funkce obecně pracují s geometriemi, jsou připraveny na to, že pro tento sloupec mohou být v řádcích tabulky hodnoty reprezentující jiný konkrétní typ. Při návrhu schématu je pak na zvážení, zda flexibilita abstraktního datového typu je potřebná, nebo je lepší typ definovat konkrétně, což s sebou nese výhody větší čitelnosti struktury dat společně s omezením rizika, že konkrétní typ bude vlivem generalizace na Geometry zaměněn za typ jiný.

Null hodnoty: pro využití prostorového indexu na sloupci s geometrickou hodnotou je podmínkou, aby sloupec byl definován jako nenulový (NOT NULL). Toto se může odrazit na návrhu schématu. Pokud bychom chtěli ukládat údaje o lokaci do jedné tabulky společně s ostatními atributy entity, pak v případě že by pro nějaký záznam údaje o lokaci chyběly, nemohli bychom uložit ani ostatní existující atributy.

Souřadný systém: pro využití souřadných systému je nutné při definici sloupce určit identifikátor SRID zvoleného systému. Data geometrie v jiném souřadném systému nebude umožněno do tohoto sloupce vložit.

3.8.2 MariaDB

Základní podpora prostorových funkcí v MariaDB vychází opět ze specifikace OGC SFA, data lze opět indexovat pomocí struktury R-Tree a pro práci s prostorovými daty je k dispozici většina funkcí, jejichž signatury a chování jsou obdobné, jako v případě MySQL.

Bohužel ani v aktuální verzi 10.5.9 (ke dni 31.3.2021) podpora některých zásadních funkcí není na takové úrovni, jako v MySQL. Zmiňme například nepřítomnost `ST_Distance_Sphere`, tedy funkce pro zjištění vzdálenosti mezi objekty, která při výpočtu zohledňuje zakřivení zemského povrchu [39].

Podpora souřadných referenčních systémů je nejasná. Ve strohé dokumentaci se například v popisech funkcí pro vytvoření prostorových objektů žádný parametr pro volbu identifikátoru souřadného systému nevyskytuje. Stejně tak je obtížné dohledat, která z funkcí využije během vykonávání prostorový index.

3.8.3 PostGIS

Text této kapitoly čerpá z oficiální dokumentace PostGIS [40] [23].

PostGIS je rozšíření relační databáze PostgreSQL. Jde o rozšíření externí, není implicitně součástí PostgreSQL. Podobně jako MySQL, PostGIS implementuje prostorové rozšíření dle specifikace OGC SFA.

Podporovaných souřadných referenčních systémů je v aktuální (ke dni 31.1.2021) verzi 3.1.1 více než 3000. PostGIS používá knihovnu Proj4 pro podporu funkcí pro transformaci souřadnic. Údaje pro konverze jsou uloženy ve sloupci `PROJ4TEXT` v tabulce `SPATIAL_REF_SYS` spolu s ostatními atributy souřadného systému.

Datové typy

PostGIS podporuje nejen základní prostorové datové typy jako MySQL, ale i ty zbývající ze specifikace OGC SFA (např. Curve). Prostorová data lze zadávat a získávat ve formátech WKT a WKT. Existují však funkce podporující i jiné formáty, jako GeoHash, GML, GeoJSON a další.

Při práci s prostorovými datovými typy je však v PostGIS vždy nutné ještě zvolit, zda budou reprezentovány jako geometrické (geometry) či geografické (geography). Geometry se používá za

předpokladu, že prostorová data jsou definována v rovině, kdežto u geography se předpokládá, že souřadnice prostorových dat jsou definovány pomocí zeměpisné šířky a délky a pocházejí tedy z elipsoidu představujícího povrch Země. Výpočty na kouli či elipsoidu jsou náročnější než výpočty v kartézské soustavě souřadnic [41]. Jako příklad uveďme výpočet vzdálenosti mezi 2 body, kde se v případě geography počítá vzdálenost vlivem zakřivení Země jako oblouk, kdežto v kartézském systému jako linie. Přetypování mezi oběma typy je možné.

Při volbě prostorového datového typu záleží na tom, k čemu jsou data určená, jaké operace nad nimi budeme provádět a v jakém souřadném systému jsou zadaná. Máme-li souřadnice v geografickém systému (např. WGS84) a pokud nám postačí funkce s nativní podporou typu geography, zvažme použití tohoto typu. Výsledky budou přesnější a vykonávání nativních geography funkcí bude rychlejší, jelikož nebude nutné přetypovávat a konvertovat data z geometry na geography. Ilustrujme tvrzení na funkci `ST_Distance` pro výpočet vzdálenosti mezi dvěma body, které jsou zadané v geografickém systému WGS84. Pro typ geometry tato funkce očekává souřadnice v kartézském systému a pro souřadnice v systému WGS84 nevrátí hodnotu v metrech, ale ve stupních. Pro geografické souřadné systémy musíme tedy provést buď konverzi z geometry na geography, nebo volat `ST_DistanceSphere`, která provádí výpočet na kouli a tedy i zde dochází ke konverzi. Oproti tomu, pokud máme data v typu geography, vrací `ST_Distance` výsledek v metrech a funkce bude tedy rychlejší než pro typ geometry, protože odpadá nutnost explicitní konverze do geography.

S přihlédnutím k cíli této práce lze říci, že oba typy jsou použitelné bez výraznějších rozdílů. Upřednostnili bychom geography, jelikož pro implementaci hledání míst do určité vzdálenosti od výchozí lokace nám postačí funkce `ST_Distance` či `ST_DWithin`, které obě podporují tento typ bez nutnosti přetypování na geometry. Navíc očekáváme, že data lokací budou v geografickém souřadném systému WGS84.

Indexy

Při volbě prostorového indexu v PostGIS je možné vybírat z těchto typů: GiST, BRIN, SP-GiST. Zmiňme jen typ GiST, který vychází ze struktury, kterou jsme si již představili. GiST je generalizací různých implementací vyhledávacích stromů (odtud jeho název Generalized Search Tree), která je dále rozšiřitelná dle vlastností dat, která se mají indexovat. Pro prostorová data používá PostGIS R-Tree strukturu implementovanou jako rozšíření GiST.

Vybrané funkce

Podobně jako u MySQL platí podmínka shodných souřadných systému pro různá prostorová data na vstupu a výstupu funkcí.

Všechny funkce s nativní podporou typu geography provádějící výpočty vzdáleností, umožňují pomocí posledního parametru nahradit výchozí způsob výpočtu na elipsoidu za výpočet na kouli, což

se projeví zhoršením přesnosti výsledku, ale urychlením vykonávání funkce. Záleží na konkrétních porovnávaných lokací, k jak velké chybě přesnosti dojde.

Opět se soustřeďme na některé funkce prostorové analýzy, které by našly uplatnění při řešení této práce.

- `ST_Distance(g1, g2, s)`: funkce vrací vzdálenost (v metrech) mezi zadanými objekty. Funkce nepoužívá prostorový index, protože nepracuje s obálkou (MBR). Pokud bychom chtěli hledat všechny body do určité vzdálenost od výchozího bodu, dotaz by funkci volal nad všemi záznamy tabulky.
- `ST_Buffer(g, d)`: na rozdíl od MySQL, v PostGIS tato funkce podporuje různé souřadné systémy. Nicméně pro nalezení bodů, které jsou ve vzdálenosti menší nebo rovné hodnotě d, je doporučováno použití funkce `ST_DWithin`.
- `ST_DWithin(g1, g2, d, s)`: funkce vrací booleovskou hodnotu dle toho, zda je objekt `g2` vzdálen maximálně `d` metrů od objektu `g1`. Pokud je k dispozici, může být použit prostorový index. Funkce využívá ohraničujícího obdélníku kolem výchozího bodu a pak hledá prvky, které svými obálkami tuto hranici přesahují. Tím se redukuje počet záznamů, pro které je třeba zjišťovat přesnou vzdálenost od výchozího bodu.
- `ST_Transform(g, srid)`: funkce umožní transformovat souřadnice mezi souřadnými systémy. V případě použití systému PostGIS by bylo možné takto transformovat i data ze systému S-JTSK do WGS84. Ukázka kódu: 3.3.

```
SELECT ST_Transform(
  'SRID=5221;POINT(-515561.9581887145 -1166539.88296505)'::geometry,
  4326);
```

Listing 3.3: Ukázka transformačního dotazu

Operátory

Pro sestavení dotazu je možné využít operátorů, které nabízí PostGIS pro práci s prostorovými daty. Většina operátorů pracuje s MBR objektů, což jim umožňuje využít R-Tree index, pokud je k dispozici. Zároveň mohou být vrácené výsledky nepřesné, jak vyplývá z podstaty ohraničujících obdélníků. Například operátor && vrací hodnotu pravda i v případech, kdy se objekty nijak neprotínají, ale jejich MBR ano.

Návrh schématu

Při návrhu a tvorbě databázového schématu je nutné zvážit následující možnosti:

Typ prostorových dat: pro vytvoření sloupců s prostorovými datovými typy je nutné zvolit, zda jde o data typu geometry, či geography, jak už bylo uvedeno výše.

Null hodnoty: na rozdíl od MySQL, PostGIS nevyžaduje pro využití prostorového indexu na sloupci s prostorovými daty, aby sloupec byl definován jako „not null“ a to díky použitému indexu R-Tree-over-GiST, který může zaindexovat i sloupce obsahující null hodnoty [42] (je „null-safe“).

Souřadný systém: pro využití souřadných systému je nutné při definici sloupce určit identifikátor SRID zvoleného systému.

3.8.4 MongoDB

Text této kapitoly čerpá z oficiální dokumentace MongoDB [43].

MongoDB je dokumentová NoSQL databáze, organizující data do kolekcí dokumentů ve formátu BSON (binární reprezentace komplexních datových struktur podobných formátu JSON). Databáze pro prostorové dotazy používá souřadný systém WGS84.

Datové typy

Podpora prostorových datových typů vychází z typů definovaných ve formátu GeoJSON a jsou totožné s těmi dostupnými v MySQL. GeoJSON je výchozí formát pro reprezentaci prostorových dat v MongoDB. Objekt GeoJSON (v terminologii MongoDB se takový objekt nazývá Document a jeho analogií v relačních databázích je záznam v tabulce) bude obsahovat atributy `type` pro specifikaci prostorového datového typu a `coordinates` pro zadání souřadnic (v pořadí *zeměpisná délka, šířka*). Při tomto zadání pak výpočty nad daty zohledňují kulovitý tvar Země a předpokládají, že lokace je zadaná v souřadném systému WGS84. Žádné jiné systémy implicitně podporovány nejsou.

Souřadnice bodu lze zadat i jako dvouprvkové pole o dvou číselných hodnotách, ale v tom případě lze provádět nad daty analýzu pouze v euklidovské rovině. Tento přístup je v MongoDB dokumentaci označen za zastaralý (legacy), nebudeme se mu tedy dále věnovat.

Indexy

Struktura 2d index slouží pro indexaci dat ve dvourozměrné rovině. Tento typ indexu však nebere v potaz zakřivení Země a jeho použití může vést u některých prostorových objektů k nepřesným výsledkům.

Pro zeměpisné souřadnice je vhodnější použít index 2dsphere, který z indexu 2d vychází. Tento index podporuje null hodnoty v indexovaných atributech. Takovéto záznamy jsou prostě indexem ignorovány.

MongoDB nejdříve pro každý objekt vypočítá kód (variantu geohash kódu, viz kapitola 3.6.3), a to na principu rekurzivního dělení celkového prostoru do kvadrantů. Levý spodní kvadrant je označen hodnotou 00 a ostatní kvadranty budou ve směru hodinových ručiček označeny hodnotami 01, 10 a 11. Při zanoření, za účelem rozdělení samotných kvadrantů, budou vnitřní kvadranty označeny

stejným způsobem, avšak jejich kódy budou vždy začínat kódem rodičovského kvadrantu. Výsledná hodnota po zřetězení kódů vedoucích k poslední zanořené buňce, je kódem objektu. Tato hodnota je pak zaindexována do B-stromu.

Bitovou přesnost velikosti binární reprezentace bodu v prostoru lze pro index parametrizovat. Větší přesností lze urychlit vyhledávání v prostorových datech na úkor rychlosti vkládání a zvýšenými požadavky na prostor pro uložení dat.

MongoDB poskytuje ještě třetí typ indexu pro prostorová data, a to sice index geoHaystack. Jelikož je v současné verzi systému označen za zastaralý, nebudeme se jim blíže zabývat.

Funkce

Pro práci s prostorovými daty nabízí MongoDB následující operátory:

- `$geoIntersects`: viz metoda `Intersects` specifikace OGC SFA. Může využít 2dsphere index.
- `$geoWithin`: viz metoda `Within` specifikace OGC SFA. Může využít 2d a 2dsphere indexy.
- `$near`: operátor vrací kolekci objektů které leží v zadané vzdálenosti od zadaného výchozího prostorového objektu. Vzdálenost lze definovat pomocí atributů `minDistance` a `maxDistance` pro vymezení minimální a maximální vzdálenosti. Výstupní kolekce bude setříděná vzestupně dle vzdálenosti. Pro použití funkce musí být nad daty definován prostorový index 2d, resp. 2dsphere (pro rovinu resp. pro geografické souřadnice).
- `$nearSphere`: operátor funguje na stejném principu, jako `$near`, s tím rozdílem, že výpočty jsou vždy sférické. Výsledek bude přesnější i v případě souřadnic uložených v rovině a indexu typu 2d. Operátor `$nearSphere` můžeme použít pro řešení jednoho z problému této práce – nalezení lokací v určitém okolí kolem nějakého bodu. Jiná varianta by byla kombinace funkcí `$geoWithin` s objektem definovaným jako `$centerSphere` – tedy s kruhem o nějakém poloměru se středem v zadaném bodě.

3.8.5 Elasticsearch

Text této kapitoly čerpá z oficiální dokumentace Elasticsearch [44].

Elasticsearch je dalším databázovým systémem z rodiny NoSQL databází. Jedná se o open source systém zaměřený na vyhledávání obsahu a jeho analýzu.

Indexy

Pro indexaci dat typu geo_shape (viz níže) Elasticsearch prostorové objekty nejdříve rozloží na trojúhelníkovou síť a každý z trojúhelníků je zaindexován do struktury Bkd-Tree [45].

Datové typy

Typ geo_point slouží pro zadání souřadnic (v pořadí zeměpisná šířka a délka) nějakého bodu. Lze zadat přímo souřadnice, jejich Geohash kód nebo definici objektu Point pomocí WKT formátu.

Pro reprezentaci složitějších tvarů, než je bod, slouží datový typ geo_shape. Pomocí něj lze definovat konkrétní prostorové typy, které vychází z formátu GeoJSON a jsou totožné s těmi, které jsme zmínili v kapitole o MySQL.

Tyto objekty lze definovat pomocí formátů WKT a GeoJSON. Navíc jsou k dispozici typy Envelope (minimální ohraničující obdélník) a Circle (kruh) zadaný středovým bodem a poloměrem.

Elasticsearch pracuje se všemi prostorovými souřadnicemi pouze v souřadném systému WGS84.

Funkce

- `geo_bounding_box`: pomocí tohoto dotazu lze vyhledat objekty typů geo_point a geo_shape, které protínají oblast definovanou ohraničujícím obdélníkem. Pro hledané objekty obsahující nějakou hranu, je vztah průniku zřejmý. Objekty typu bod budou nalezeny, pokud leží uvnitř ohraničující oblasti nebo přímo na její hranici. Ohraničující obdélník je definován párem souřadnic určující diagonálně protilehlé rohy obdélníku.

- `geo_distance`: dotaz nalezne objekty do určité zadané vzdálenosti od nějaké lokace typu geo_point. Vzdálenost lze určit řadou metrických a imperiálních jednotek. Výpočet vzdálenosti bere v potaz zakřivení Země. Pro rychlejší a zároveň nepřesnější výsledek lze způsob výpočtu změnit parametrem `distance_type` na rovinný.

- `geo_polygon`: na rozdíl od `get_bounding_box`, zde můžeme definovat ohraničující oblast obecněji jako libovolný validní polygon, ne jen jako obdélník. Budou vráceny jen ty objekty, které zcela leží uvnitř této oblasti.

- `geo_shape`: dotaz opět slouží pro hledání výsledků v určitém vztahu vzhledem k výchozímu objektu. Výchozí ohraničující objekt lze definovat referencí na již existující indexovaný objekt. Parametr `strategy` slouží pro definici vztahu mezi výchozí oblastí a kandidáty, a jsou podporovány tyto relace: Intersects, Disjoint, Within, Contains.

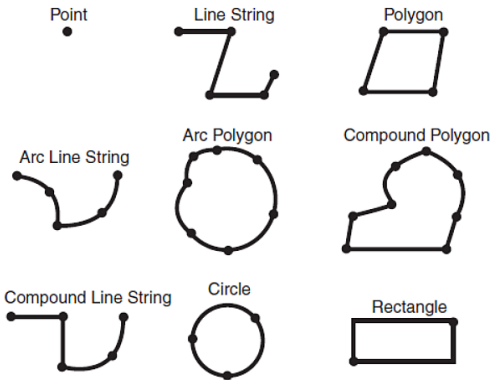
V případě objektů typu geo_point lze hledat pouze ty, které jsou s výchozím objektem v relaci Intersects. Pro hledané objekty geo_shape jsou podporované všechny uvedené relace.

3.8.6 Oracle, Microsoft SQL Server a IBM Db2

Text práce se soustředí na produkty, které jsou volně k použití i pro komerční účely. Popišme alespoň okrajově tři nejpopulárnější komerční databázové systémy společností Oracle, Microsoft a IBM z pohledu prostorové podpory.

Podpora prostorových dat a funkcí je ve všech třech systémech na srovnatelné úrovni, alespoň co se týče základní funkcionality, které jsme se věnovali v předchozích kapitolách. V Oracle se problematice věnuje komponenta Oracle Spatial, v Db2 zase modul Spatial Extender. V Microsoft SQL Server (MSSQL) nemá prostorové rozšíření žádné konkrétní označení.

Dostupné datové typy a funkce vycházejí ze specifikace OGC SFA. Oracle a MSSQL navíc nabízí typy jako Arc Polygon či Curve Polygon pro objekty, kde spojnicí mezi vrcholy může být i oblouk a ne jen linie, viz 3.7. V případě MSSQL je při definici prostorového objektu a při definici prostorového sloupce tabulky navíc nutné zvolit mezi typem geometry (pro rovinu) a geography (pro reprezentaci dat na elipsoidu), podobně jak je to tomu v PostGIS. Pro reprezentaci prostorových dat, ať už jako



Obrázek 3.7: Ukázka některých prostorových datových typů dostupných v Oracle [46]

jejich vstup nebo výstup, jsou podporovány formáty WKT, WKB a GML. Oracle podporuje navíc GeoJSON, DB2 zase formát ESRI Shape.

Je podporována řada souřadných referenčních systémů a s tím spojené přesnější výpočty reálných vzdáleností na elipsoidu. Pro systém WGS84 používá Oracle identifikátor SRID 8307, oproti běžnému kódu 4326. V MSSQL je oproti ostatním systémům možné do sloupce s prostorovým datovým typem uložit objekty definované v různých souřadných systémech, ale platí pravidlo, že objekty, které jsou vzájemně porovnávány, nebo vstupují jako parametry do nějaké funkce zjišťující jejich prostorovou relaci, musí mít shodné SRID, jinak funkce vrátí null hodnotu.

Pro indexaci prostorových dat využívá Oracle strukturu R-Tree. MSSQL a DB2 nabízejí index typu Grid s maximálními počty úrovní 4, resp. 3 (u DB2 je možnost zvolit mezi jednou až třemi úrovněmi při počáteční definici indexu).

Použití prostorového indexu je u všech těchto systémů podporované minimálně ve funkcích pro zjištění průniku či shodnosti prostorových objektů a pro určování vzdáleností mezi objekty (Intersects, Equals, Distance).

3.8.7 Srovnání MySQL a PostGIS

Srovnejme výkon dvou popsaných relačních systému řízení báze dat: MySQL a PostGIS. Systém MariaDB, pro jeho nedostatečnou podporu prostorových funkcí, do srovnání nebudeme zahrnovat.

Budeme vykonávat dotaz pro nalezení lokací v oblastí 10 km od nějaké výchozí polohy. V obou DB systémech provedeme hledání s využitím ohraničujícího obdélníku a bez. Pomocí MBR více omezíme prohledávanou množinu dat, a to s použitím prostorového indexu. Ve variantě bez obálky musíme porovnat vzdálenost výchozího bodu s každým jiným bodem v celé tabulce. Prostorový index v tomto případě nebude využit.

Nejdříve budeme vyhledávat mezi 4000 prodejci. Tím je tedy vyhledávací prostor poměrně omezen a odpovídá reálnému použití aplikace. Následně budeme vyhledávat v množině 2,9 miliónů záznamů poloh. Srovnáme časy vykonávání dotazů na vývojářské sestavě[2].

Tabulka 3.1: Přehled průměrných časů vykonávání dotazů na vývojářské sestavě

	4 tisíce záznamů		2,9 miliónů záznamů	
SŘBD	1. bez MBR	2. MBR	3. bez MBR	4. MBR
MySQL	0,2 s	3,4 s	153 s	3,4 s
PostGIS	0,2 s	0,2 s	15 s	0,2 s

1) Bez prostorové obálky je dotaz nad malým počtem dat vykonán okamžitě v obou systémech. Prostorový index využit nebyl, což při malém množství dat nebylo překážkou. Doba vykonávání je velice nízká a totožná v obou systémech.

2) S použitím obálky nad malým počtem dat se vykonávání dotazu v MySQL prodlužuje na 3,4 sekundy, byť je patrné využití prostorového indexu (`idx_vfr_loc_g` nad `VFR_LOC(G)`) a plán vykonání dotazu je lepší než v předchozím případě. Pro malé množství dat by tedy v případě použití MySQL bylo výhodnější zvolit verzi dotazu bez použití MBR. V případě PostGIS je však čas vykonávání dotazu stejný, jako bez použití indexu. V tomto případě je tedy zřejmé, že PostGIS je rychlejší než MySQL.

3) V případě většího množství dat jsou již rozdíly mezi porovnávanými systémy mnohem větší. Ve verzi dotazu bez prostorového indexu není MySQL schopen vrátit výsledek v rozumné době (nameřený čas byl 2 minuty a 30 sekund). Naproti tomu PostGIS vrátil výsledek po 15 sekundách a to bez použití indexu na téměř 3 miliónech záznamů.

4) V posledním variantě dotazu s MBR byla odpověď MySQL dostupná za stejnou dobu, jako v druhém případě. Při použití indexu neměl počet prohledávaných záznamů žádný dopad na rychlost vrácení výsledku. Totéž se dá říct o PostGIS, ale s tím podstatným rozdílem, že i v případě 2,9 miliónů záznamů vrací PostGIS výsledky do doby 0,2 sekund.

[2]HW konfigurace: Intel(R) Core(TM) i5-7300U CPU @ 2.60GHz, 16 GB RAM, SSD disk.
SW vybavení: Windows 10 64 bit, MySQL 8.0.17, PostGIS 3.0.1, PostgreSQL 12.2.

Z toho srovnání vyplývá, že systém PostGIS je výkonnostně efektivnější, než systém MySQL. Pokud budeme prohledávat v tabulce s prostorovými body o velikosti v řádech tisíců, pak v případě MySQL je lepší konstruovat dotaz tak, aby nevyužíval prostorový index.

Podpora prostorových funkcí systému PostGIS je rozsáhlejší, než u systému MySQL, pro který platí, že ani v aktuální (ke dni 31.3.2021) verzi 8.0.23 mnohé jeho funkce stále nepodporují souřadné referenční systémy a nelze je tedy použít pro přesné kalkulace nad zeměpisnými souřadnicemi.

MySQL dotazy pro případy 3 a 4 jsou k dispozici v příloze D na stránce 98. PostGIS dotazy nejsou uvedeny, jelikož se téměř neliší. Pouze by v nich byla nahrazena funkce `MBRContains` za funkce `ST_Contains` a `ST_Envelope`.

Kapitola 4

Nástroje třetích stran a cloudových aplikací

V této kapitole budou představeny některé aplikace třetích stran, které lze použít k dosažení cílů představených v úvodních kapitolách textu. Zaměříme se nyní spíše na teoretickou stránku věci a praktickému použití se budeme věnovat až v kapitole 6 popisující implementaci.

4.1 Mapové internetové aplikace

V oblasti aplikací pracujících se zeměpisnými prostorovými údaji je záhodno nabídnout vizualizace dat na konkrétním mapovém podkladu. Internetové mapové aplikace rovněž mohou posloužit jako zdroje dat, např. pro vyhledání souřadnic pro určité adresy či pro získání navigačních údajů při plánování tras mezi objekty. Použití funkcí dostupných internetových mapových aplikací pomocí základního uživatelského rozhraní v okně webového prohlížeče určeného pro běžného uživatele nebývá nijak omezováno. Někteří poskytovatelé těchto aplikací nabízejí i API rozhraní, jehož použití už může být vymezeno různými podmínkami.

Představíme si následující aplikace: Google Maps (nejpopulárnější komerční mapová aplikace dle výzkumu z roku 2018 [47]), OpenStreetMap (zástupce „otevřené“ aplikace) a Mapy.cz (česká mapová aplikace poskytující také údaje a funkce vztažené k České republice, které nejsou dostupné v ostatních zmíněných aplikací).

Všechny tři aplikace umožňují alespoň následující funkce: prohlížení světové mapy, zobrazení lokace dle zadané adresy či dle geografických souřadnic, nalezení silniční trasy alespoň mezi 2 body a možnost do jisté míry parametrizovat URL mapové aplikace pro zobrazení nějaké výchozí polohy. Aplikace nabízejí další specifické funkce pomocí API rozhraní, jehož použití může být zpoplatněno.

Pro projekci mapy využívají tyto aplikace kartografické zobrazení WGS 84/Pseudo-Mercator a pro práci se souřadnicemi je podporován systém WGS84 [3] [4].

4.1.1 Google Maps

Aplikace Google Maps (http://maps.google.com) je volně dostupný komerční software.

Pro projekci mapy využívá aplikace od roku 2005 kartografické zobrazení WGS 84/Pseudo-Mercator, ale od roku 2018 kvůli velkém zkreslení v oblastech v blízkosti polárních oblastí, Google Maps primárně zobrazuje data přímo na trojrozměrném glóbu. Mezi zobrazeními lze plynule přepínat až do blízkého přiblížení, kde oblast viditelná v okně prohlížeče odpovídá oblasti přibližně 40 až 100 km široké. Při tomto a bližším přiblížení je použito pouze zobrazení Pseudo-Mercator, které nelze změnit.

4.1.2 Mapy.cz

Aplikace Mapy.cz společnosti Seznam.cz poskytuje obdobné funkce, jako aplikace Google Maps. Jde rovněž o volně dostupnou (http://www.mapy.cz), avšak komerční aplikaci.

Oproti konkurenčním webovým mapovým aplikacím nabízí Mapy.cz funkce specifické pro území České republiky. Existuje zde více mapových vrstev (turistická, zimní, letecké z několika období mezi lety 2003 až po současnost, historická mapa z 19. století a jiné), možnost snadného přesměrování na veřejně dostupné katastrální informace či možnost vyhledávání pomocí souřadnic v systému S-JTSK.

4.1.3 OpenStreetMap

Aplikace OpenStreetMap (OSM), dostupná na adrese https://www.openstreetmap.org, je součástí stejnojmenné iniciativy s cílem sestavit a poskytovat editovatelné mapy zdarma. Jde o kolaborativní dílo, otevřené veřejnosti, která se může zapojit do kolekce a sestavování mapových a jiných geografických podkladů. Licence OSM je typu Open Database License, umožňující volné použití licencovaného díla s podmínkou, že tato volnost zůstane zachována i pro následně odvozená díla.

OSM poskytuje dva typy aplikačního programového rozhraní. Primární API je založeno na REST architektuře a slouží hlavně pro podporu editace a přispívání do kolektivního mapového díla. Overpass API nabízí základní funkce pro čtení OSM dat. Součástí dokumentace OSM je i seznam knihoven, které do nějaké míry využívají OSM data pro poskytnutí dodatečné a specializované funkcionality, jako je například podpora navigace, zobrazení části mapy v rámci webové stránky nějaké aplikace, vyhledávání souřadnic a další. Knihovny se od sebe dále liší podporou programovacích jazyků a platforem a specifickými licenčními podmínkami [48].

Pro řešení zadání této práce navrhujeme využití knihoven Leaflet a Open Source Routing Machine (OSRM).

4.2 Leaflet

Leaflet je knihovna pro jazyk Javascript, která umožňuje vložit mapu do webové stránky jako vnořený element. Použití knihovny je zdarma.

Mapa je interaktivní (pokud toto není schválně potlačeno v konfiguraci instance mapy), její obsah lze tedy posouvat, přibližovat a oddalovat. Knihovna podporuje použití i v prohlížečích mobilních zařízeních, kde lze s mapou pracovat pomocí běžných gest pro dotyková zařízení (například přiblížení mapy pomocí dvou prstů, tzv. pinch-to-zoom).

Výchozí rozměry mapového okna, úroveň přiblížení, vycentrování na určitou lokaci a jiné možnosti jsou plně konfigurovatelné. Lze konfigurovat funkce pro zpracování událostí, které se volají například při uživatelově kliknutí myší do oblasti mapy.

Dále je umožněno do mapy vkládat značky, které mohou obsahovat nějaký popisek, či vrstvy, které mohou být využity například pro definování a označení nějaké oblasti. Vzhledem k zadání této práce se nabízí využití těchto funkcí pro označení výsledků vyhledávání na vložené mapě, kdy značky budou reprezentovat konkrétní nalezené prodejny či prodejce a vrstvou sestávající ze spojených úseček můžeme vizualizovat trasu mezi těmito výsledky.

Primárním formátem pro zadání prostorových dat, například pro definování vrstev, je GeoJSON. Podporu dalších formátů lze získat pomocí rozšíření, kterých je velké množství (dokumentace hovoří o stovkách [49]). Poskytují i funkce z oblastí vyhledávání souřadnic, navigace a jiných.

4.2.1 Výběr mapových dat

Při nastavení knihovny je nutné zvolit jeden ze dvou formátů služeb poskytujících mapová data. Knihovna Leaflet podporuje formáty Tile Map Service (TMS), Web Map Service (WMS) a Web Map Tile Service (WMTS). TMS bylo vytvořeno společností Open Source Geospatial Foundation, WMS a WMTS společností Open Geospatial Consortium.

Rozhraní formátu TMS je definováno jako REST API a webová služba poskytující TMS data vrací, dle parametrů (souřadnice, úroveň přiblížení) předaných v URL, rastrová data s příslušným výřezem mapy. Na stejném principu funguje v Leaflet i konfigurace WMTS.

Definice rozhraní WMS zahrnuje dvě povinné funkce: GetCapabilities, která vrací informace o parametrech konkrétní služby a GetMap, pomocí které se získá požadovaný výsek mapy. Leaflet pracuje pouze s funkcí GetMap a protože ta je dle specifikace povinná, tak při konfiguraci URL WMS služby není třeba název této funkce vůbec zadávat, nicméně je nutné ještě specifikovat pomocí parametru název mapové vrstvy poskytované WMS serverem [50].

4.3 Open Source Routing Machine

Text této kapitoly čerpá z oficiální dokumentace OSRM [51].

Aplikace Open Source Routing Machine (OSRM), poskytuje funkce pro plánování tras po silničních sítích. Aplikaci lze používat zdarma. Pro směrovací algoritmy používá jako vstupní data podklady z projektu OpenStreetMap. Ty mohou být zcela importovány lokálně, například na server, na kterém je spuštěna lokální instance knihovny. Funkčnost aplikace je pak s takto dostupnými daty nezávislá na internetovém připojení, či na responzivitě aplikace běžící na jiném serveru.

Funkce aplikace lze použít dvěma způsoby: přes HTTP rozhraní, nebo pomocí knihovny libosrm pro jazyk C++. Druhý způsob je efektivnější, pokud chceme OSRM volat právě z prostředí C++. Jelikož naše zamýšlená cílová demonstrační aplikace je webovou aplikací, budeme se dál v popisu věnovat jen prvnímu přístupu.

Funkce aplikace voláme pomocí URL adresy, kterou předáváme specifické parametry volané služby. Pro všechny obecné požadavky existuje několik generických parametrů. Jsou jimi například:

- profile: pro zadání způsobu dopravy (autem, na kole, pěšky), kterou má volaná funkce zohlednit;
- coordinates: parametr umožní předat zeměpisné souřadnice (*zeměpisná délka* a *šířka* oddělené čárkou). Souřadnice lze zadat jako množinu, kde dvojice souřadnic jsou oddělené středníkem. Dalším podporovaným formátem pro zadání souřadnic je polyline;
- format: určuje výstupní formát dat. Výchozí hodnota je JSON a dalším podporovaným formátem je flatbuffers.

Zpráva odpovědi (ve formátu zvoleném pomocí parametru `format`) vrací také kromě samotného výsledku dotazu i několik generických parametrů. Součástí každé odpovědi bude parametr `code` popisující stav výsledku volání. Jeho hodnota bude „Ok“ pokud nedošlo k chybě. V opačném případě bude hodnota odpovídat kódu popisující důvod chyby (např. „InvalidUrl“, „InvalidQuery“). Další výstupním parametrem, byť nepovinným, je message ve které může být popis stavu výsledku a případně i bližší popis chyby, pokud k nějaké došlo.

OSRM nabízí následující funkce:

- Route
- Table
- Trip
- Match
- Nearest
- Tile

4.3.1 Hledání nejrychlejší trasy

Route

Funkce Route se pokusí nalézt nejrychlejší, ne nutně nejkratší, trasu mezi zadanými souřadnicemi. V případě, že je dvojic souřadnic tři a více, funkce bude respektovat zadané pořadí a bude hledat trasy mezi dvojicemi, jak jdou za sebou. Tedy nejdříve mezi prvním a druhým bodem, následně mezi druhým a třetím atd.

OSRM pro hledání nejkratších tras používá metodu využití hierarchií silniční sítě. Hierarchií je myšleno, že některé segmenty silniční sítě urychlí průjezdy mezi větším počtem různých lokací (např. dálnice) než jiné (např. slepá ulice) a jsou tak hierarchicky na vyšší úrovní. Takový algoritmus pracuje s daty, které jsou vytvořené v přípravné fázi, kdy se právě takovéto segmenty na hierarchicky vyšších úrovních vyhledají.

Nad předzpracovanými daty je pak volán Dijkstrův algoritmus pro nalezení nejkratší trasy. Algoritmus funguje na principu hledání nejkratší vzdálenosti [52]:

1. Všechny body grafu se přiřadí do množiny nenavštívených bodů.
2. Každému bodu se nastaví hodnota předběžné vzdálenosti: výchozímu bodu se nastaví na nulu, všem ostatním na nekonečno.
3. Pro výchozí bod se vypočítají vzdálenosti ke všem dosud nenavštíveným sousedům (body spojené jednou hranou). Ty se porovnají s předběžnými vzdálenostmi těchto nenavštívených bodů. U bodů, kde je nalezená vzdálenost menší, uloží se tato jako předběžná vzdálenost bodu.
4. Výchozí bod se odstraní z množiny nenavštívených bodů.
5. Algoritmus skončí:
   (a) pokud cílový bod byl nastaven jako nový výchozí bod. V této chvíli je už cena (nejkratší) cesty z výchozího bodu do cílového známa;
   (b) nebo pokud všechny nenavštívené body jsou z výchozího bodu nedosažitelné. Nejkratší předběžná vzdálenost bodů této množiny odpovídá nekonečnu, jakožto výchozí hodnotě.
6. V ostatních případech algoritmus pokračuje. Vybere se nový výchozí bod z množiny nenavštívených bodů, a to ten s nejmenší hodnotou předběžné vzdálenosti. Algoritmus pokračuje znovu od kroku 3.

V dokumentaci OSRM se uvádí, že výsledky nalezení nejkratší cesty mezi dvěma body mapy velikostně odpovídající ploše a hustotě silniční sítě Evropy, jsou nalezeny v čase kolem 1ms.

Funkce jde dále parametrizovat. Uveďme pro příklad některé parametry:

- alternatives: příznak značící, zda se mají hledat alternativní trasy. Výchozí hodnota false má za následek vrácení maximálně jedné trasy;

- geometries: tento parametr se opakuje i u jiných funkcí OSRM a určuje formát výstupních prostorových dat. Možnosti jsou: polyline, polyline6, GeoJSON;

- overview: parametr určuje míru detailů vrácených ve výsledném objektu. Funkce route bude vracet i prostorový objekt (geometry), který představuje výslednou nalezenou trasu (linii) mezi vstupními souřadnicemi. Parametr overview nastavený na hodnotu full či simplified ovlivní míru detailů vrácené linie. Vrácení tohoto prostorového objektu lze předem zcela potlačit nastavením parametru overview na hodnotu false. V takovém případě nalezneme ve výsledku pouze informace o délce trasy a očekávaném času, který cesta po této trase zabere. Tyto informace mohou být v patřičném případu užití zcela dostačující a potlačení vrácení geometrického objektu vede k vyšší rychlosti vykonání operace a menšímu objemu vrácených dat.

Funkce vrací kolekce objektů Waypoint a Route.

Objekty Waypoint představují průjezdní body a obsahují jejich souřadnice a jméno nejbližší nalezené ulice.

Objekt Route obsahuje atributy distance (celková délka trasy v metrech), duration (celkový cestovní čas v sekundách), kolekci legs s obdobnými atributy pro jednotlivé segmenty trasy a objekt geometry představující křivku slouženou ze souřadnic průjezdních míst. Objektů Route může být vráceno v kolekci více a představují alternativní nalezené trasy, seřazené od nejrychlejších.

Table

Služba Table, podobně jako Route, hledá nejrychlejší trasy, ale tentokrát mezi všemi možnými dvojicemi zadaných souřadnic, pokud není určeno pomocí parametru sources (resp. destinations), že některé ze souřadnic mají být pouze výchozí (resp. pouze cílové) lokace. Do parametrů sources a destinations se vkládá řetězec čísel, které odpovídají polohám souřadnic ve vstupní kolekci. Tyto indexy jsou odděleny středníkem.

Funkce je schopna vracet ve výstupním objektu dvě matice: durations (čas tras v sekundách), distances (délka tras v metrech), výchozí nastavení však vrací pouze první matici. Matice se vzdálenostmi v metrech je nutné pro výsledek aktivovat parameterem annotations nastaveným na hodnoty „distance“ nebo „duration,distance“.

Funkce používá stejný algoritmus pro nalezení trasy, jako funkce Route. Je vhodná pro hromadné zjištění délky a času trasy, např. mezi jedním vstupním bodem a velkým množstvím cílových poloh.

4.3.2 Hledání okružní trasy pro navštívení více míst

Trip

Jedním z problému plánování tras je nalezení optimálního pořadí navštívení zadaných lokací, pokud je hlavním kritériem trasy právě její celková optimalizace a na pořadí průjezdních bodů nezáleží. Jde tedy o problém obchodního cestujícího (travelling salesman problem, „TSP“). Jde také o jeden z úkolů, kterým se zabývá tato práce, a to sice poskytnutí uživateli návrh optimální trasy pro navštívení nalezených prodejen.

Jelikož je TSP NP-úplný problém, jeho implementace v rámci funkce Trip používá pro nalezení výsledku hladovou heuristiku. Hladová heuristika může v případě TSP fungovat způsobem, kdy v každém nalezeném místě algoritmus hledá další nejbližší nenavštívené místo. Nalezená trasa nemusí být ta nejoptimálnější, ale měla by být dostatečně přesná a je také zaručeno, že algoritmus skončí v přijatelném čase.

Funkce Trip konkrétně používá algoritmus farthest insertion [53] s průměrnou složitostí $O(n^2)$. Algoritmus funguje na principu postupného rozšiřování trasy o nejvzdálenější body [54]:

1. První bod i se zvolí náhodně.
2. Najde se bod j, který je od bodu i nejvzdálenější. Vytvoří se trasa $i - j - i$ (má dvě hrany).
3. Dále se bude hledat bod k, kterým se trasa rozšíří. Bod k musí být od všech bodů trasy nejvzdálenější.
4. Bod k je nutné začlenit do trasy a to tak, že jedna z hran trasy bude nahrazena dvěma hranami spojující nový bod s dvěma sousedními body trasy. Vybere se ta hrana, která po svém nahrazení zvětší celkovou délku trasy co nejméně.
5. Dokud existují nějaké nezačleněné body, pokračuje se znova od kroku 3, jinak algoritmus skončí.

Výstupem funkce jsou kolekce objektů Waypoint a Route obsahující informace o délkách, časech a průjezdních lokacích jednotlivých segmentů a také prostorový objekt zobrazující celkovou navrženou trasu. Obojí v míře detailů a formátu určených vstupními parametry geometries a overview. Viz. popis funkce Route. Každý objekt Waypoint navíc ještě obsahuje index odpovídající polohám vstupních souřadnic. Tím lze spárovat zadané lokace s průjezdními body a určit tak přesné pořadí, kterými je doporučováno vstupní souřadnice navštívit.

Dodejme, že navigační funkce dostupných webových mapových aplikací, např. těch zmíněných v kapitole 4.1, nenabízí řešení TSP problému a dokážou sestavit trasu s průjezdními body jen v takovém pořadí, v jakém byly zadány.

4.3.3 Ostatní funkce OSRM

Match

Účelem funkce Match je najít nejbližší silniční prvky pro zadané souřadnice. Oproti funkci Route funkce Match nehledá nejrychlejší trasu mezi body, ale může být například použita například pro rekonstrukci trasy zadané množstvím souřadnic.

Výsledná trasa může být nespojitá, například pokud je mezi nějakými body zadaná příliš velká časová prodleva, nebo pokud nelze najít přímé silniční spojení mezi některými body. Funkce také vyhodnocuje, zda některé zadané body nejsou odlehlá pozorování a tyto body jsou při vyhledávání ignorovány.

Nearest

Funkce Nearest pro zadané souřadnice vrátí název nejbližší nalezené ulice. Počet ulic, které mají být pro každou souřadnici bodu vráceny, lze zadat parametrem *number*.

Tile

Funkce vrací mapové podklady, jejichž princip byly již zmíněný v kapitole popisující aplikaci Leaflet 4.2. V případě OSRM funkce Tile jde o však data čistě vektorová a ne rastrová. Vektorová mapa je generována z dostupných dat importovaných do OSRM.

Vstupní parametry funkce jsou obdobné, jako například pro formát TMS, je tedy nutné specifikovat souřadnice a úroveň přiblížení.

Výstupní část vektorové mapy obsahuje dvě vrstvy. Vrstva *speeds* obsahuje pro jednotlivé části cest informace o povolených rychlostech a přibližný čas (v sekundách), který je potřebný pro absolvování úseku cesty v automobilu. Vrstva *turns* obsahuje informace o směru křižovatek, úhlech zatáček a odhadovaný čas (v sekundách) nutný pro projetí zatáčky křižovatky. Tyto informace lze využít pro sestavení přibližného modelu časové délky trasy.

Kapitola 5

Návrh implementace

Tato kapitola představí požadavky a návrh implementace demonstrační aplikace, která má za cíl ukázat, jako ověření konceptu, jedno z možných řešení zadání práce.

5.1 Motivace

Jedním z problémů e-commerce je logistika produktů mezi prodejcem a zákazníkem. To, kde se produkt nachází, může být pro zákazníka jedním z kritérií při vyhledávání produktů, a to zejména v následujících případech:

- vzdálenost produktu od zákazníka má vliv na cenu dopravy a proto může nakupující upřednostnit produkty v takové vzdálenosti, která cenu snižuje;
- nehledě na cenu dopravy může mít zákazník potřebu si výrobek před zakoupením nejdříve osobně prohlédnout a v tomto případě může preferovat ty produkty, které jsou blíž zákazníkově lokaci;
- existuje řada kategorií produktů, u kterých nelze dopravu k zákazníkovi realizovat a očekává se, že se o dopravu postará sám zákazník, nebo může jít i o sortiment, který je nepřenositelný a je vázán na konkrétní místo.

Těmito produkty mohou být například automobily, nábytek či reality.

E-commerce aplikace nabízející zboží tohoto typu, by měla pro jejich vyhledávání zohlednit i možnost zahrnutí parametrů lokace produktu a lokace zákazníka. Uživatel by pak mohl použít vzdálenost produktu jako jeden z parametrů vyhledávání a omezit výsledky na produkty dostupné do určité vzdálenosti (ať už vzdušné vzdálenosti, nebo délky trasy po silnicích) od vybraného místa, typicky lokace uživatele.

Nabízí se možnost vizualizovat výsledky na vložených mapách, díky čemuž získá zákazník jasnější představu o dostupnosti produktů v jeho okolí. Jako další funkci využívající informace o geografické

poloze produktů může aplikace nabídnout zákazníkovi službu v podobě doporučení trasy od výchozího bodu k poloze některého z výsledků, nebo navrhnout, v jakém pořadí navštívit jednotlivé lokace nalezených produktů tak, aby trasa byla co nejekeftivnější a délka celé okružní trasy co nejkratší.

Tato problematika není důležitá jen při pohledu na potřeby zákazníka. Uživatelem aplikace může být i samotný provozovatel výdejny či obchodu, který může díky lokačním informacím lépe monitorovat nabídku konkurence v jeho blízkosti.

Příkladem zmíněné e-commerce aplikace je portál Gloffer, který se zabývá zprostředkováním prodeje nových a ojetých vozů na území České republiky. Navrhované řešení a demonstrační aplikace budou navrženy tak, aby byly ve velké míře kompatibilní s přístupem aplikace Gloffer. Vyhledáváné lokace, se kterými budeme pracovat, budou představovat autosalony či autobazary. Souhrnně tyto entity označíme pojmem Prodejce, jakožto zprostředkovatele obchodu, což platí jak pro pobočky autosalónů s novými auty, tak pro autobazary s auty ojetými.

5.2 Případy užití

Představme si řešené problémy na konkrétních případech užití pro vyhledání automobilů.

Demonstrační implementace neposkytuje kompletní funkcionalitu pro každý z případů užití. U případů užití UC 1 až UC 6 je realizováno základní vyhledávání pomocí silniční vzdálenosti (UC 2), které by šlo v dalším rozvoji aplikace parametricky přizpůsobit pro ostatní případy užití. Pro ilustraci hlavního cíle práce, tedy vyhledávání pomocí geolokačních atributů, je toto řešení dostačující. Případy užití UC 7 a UC 8 jsou taktéž implementovány.

- UC 1: Najdi automobily do vzdušné vzdálenosti d

  Příklad: Uživatel hledá automobil Škoda Fabia, r. v. 2015, který se nachází ve vzdálenosti do 30 km od polohy uživatele.

- UC 2: Najdi automobily do silniční vzdálenosti d

  Příklad: Uživatel hledá automobil Škoda Fabia, r. v. 2015, který se nachází v silniční vzdálenost do 50 km od polohy uživatele.

- UC 3: Najdi automobily do dojezdové doby t

  Příklad: Uživatel hledá automobil Škoda Fabia, r. v. 2015, kde maximální čas trasy nutný pro dosažení automobilu, je 90 minut (za normálního provozu).

- UC 4: Najdi k nejbližších automobilů (dle vzdušné vzdálenosti)

  Příklad: Uživatel hledá maximálně 10 nejbližších (dle vzdušné vzdálenosti) automobilů Škoda Fabia, r. v. 2015.

- UC 5: Najdi k nejbližších automobilů (dle silniční vzdálenosti)

Příklad: Uživatel hledá maximálně 10 nejbližších (dle silniční vzdálenosti) automobilů Škoda Fabia, r. v. 2015.

- UC 6: Najdi k nejbližších automobilů (dle dojezdové doby)

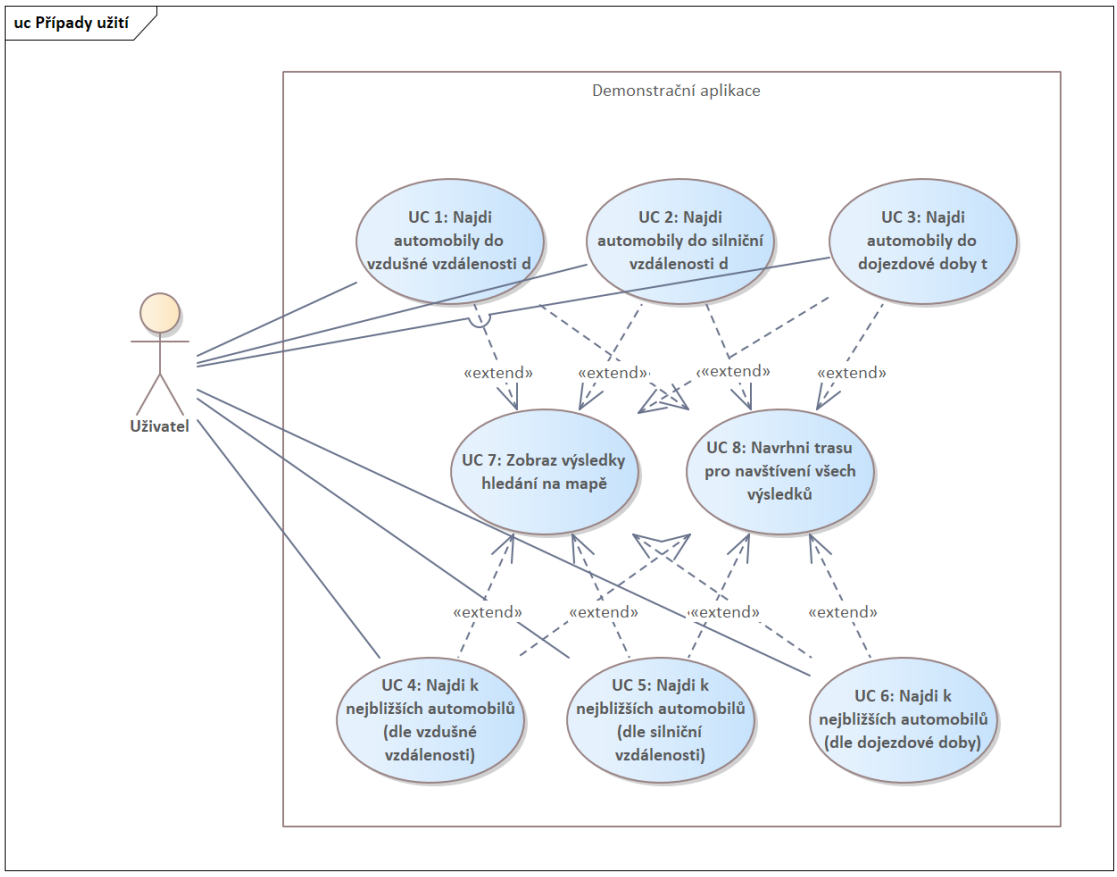
  Příklad: Uživatel hledá maximálně 10 nejbližších (dle dojezdové doby) automobilů Škoda Fabia, r. v. 2015.

- UC 7: Zobraz výsledky hledání na mapě

  Příklad: Uživatel si přeje mít výsledky vyhledávání (z UC 1 – UC 6) vizualizované na mapě vložené do stránky portálu.

- UC 8: Navrhni trasu pro navštívení všech výsledků

  Příklad: Uživatel hledá co nejoptimálnější silniční trasu pro navštívení lokací všech výsledků vyhledávání.



Obrázek 5.1: Diagram případů užití vyhledávání

Mějme na mysli, že lokací automobilu je entita Prodejce představující místo (salón, bazar atd.), kde se auto nachází. Každý z vyhledávacích případů užití lze použít i přímo pro vyhledání samotných

prodejců. Stačí tomu přizpůsobit obecné vyhledávací parametry, kde se kromě parametrů vozidel zahrnou i přímo parametry prodejců (např. typ = {salón, bazar}, značka = {Škoda, Renault, ...} a další). Tyto varianty případů užití nebudeme dále rozvádět.

Případy užití UC 1 až UC3 reprezentují situaci, kdy má uživatel představu o obecných parametrech vozidla (značka, model, rok výroby, počet najetých km, objem motoru a další), které chce koupit, ale jelikož si chce nabízené automobily osobně před nákupem prohlédnout, vybírá jen vozy, které jsou dostupné do nějaké vzdálenosti (ať už vzdušné, silniční nebo dané časem nutným k dosažení lokality). Chce mít také možnost výsledky dle těchto dodatečných kritérií třídit.

Případy užití UC 4 až UC 6 vychází z podobné situace jako u předchozích případů užití s tím rozdílem, že uživatel nechce limitovat výsledky nějakou maximální vzdáleností či časem dojezdu, ale chce vyhledat nějaký maximální počet vozidel, které jsou dle těchto kritérií seřazeny. Může se stát, že nejbližší výsledek je ve vzdálenosti (či čase), která je pro uživatele neakceptovatelná, nicméně uživatel toto zjistí rychleji, než kdyby postupně rozšiřoval parametr vzdálenosti (či času) dle UC 1 až UC 3. Jde tedy o problém kNN (k-nejbližších sousedů). Sám uživatel může aplikovat naivní řešení tohoto problému i pouze pomocí UC 1 až UC 3 a to tak, že zadá vzdálenost (či čas) natolik velkou, že tímto kritériem neomezí žádný z potenciálních výsledků. Výsledky si poté setřídí dle požadovaného parametru, kde prvních k výsledků bude představovat výsledek vyhledávání k nejbližších automobilů. Aplikace by měla poskytnout uživatelsky příjemnější způsob takového vyhledávání, který by byl i patřičně optimalizován.

Případ užití UC 7 představuje funkci, kdy jsou výsledky vyhledávání zobrazeny na vložené mapě aplikace. To uživateli umožní rychlejší vizuální orientaci v sadě výsledků. Ihned si může udělat představu o množství výsledků a o jejich polohách. Je žádoucí, aby byla tato mapa vložená přímo na stránku portálu a aby uživatel nebyl přesměrován na jinou aplikaci (např. Google Maps). Takové chování by bylo z uživatelského hlediska nepohodlné.

Případ užití UC 8 popisuje možnost uživateli navrhnout co nejideálnější trasu pro navštívení všech lokací vyhledaných vozidel. Vycházíme zde z předpokladu, že uživatel již parametrizoval vyhledávání dle svých potřeb a proto každý z nalezených automobilů je potenciální kandidát pro koupi o které rozhodne až osobní prohlídka či zkušební jízda vozidla. Pokud chce uživatel navštívit všechny nalezené prodejce, bude stát před problémem nalezení ideální trasy, která bude minimalizovat čas nutný pro její kompletní projetí.

5.3 Požadavky

5.3.1 Funkční požadavky

PROČ vyvinout demonstrační aplikaci?

- Vytvořit ověření konceptu a demonstrovat tak použitelnost a funkčnost navrhovaného řešení v praxi.

- Ověřit možnosti rozšíření aplikace Gloffer s cílem zvýšení uživatelského komfortu a lepší konkurenceschopnosti.

KDO bude s demonstrační aplikací pracovat?

- Architekt aplikace Gloffer, nebo osoby zodpovědné za zvážení a navržení implementace možného rozšíření aplikace Gloffer.

CO bude demonstrační aplikace evidovat?

- Všechny oficiální poštovní adresy ČR (získané z číselníku RÚIAN).
- GPS souřadnice většiny (98,95%) poštovních adres.
- Poštovní adresy některých skutečných prodejců automobilů na území ČR

.

5.3.2 Nefunkční požadavky

- Pro implementaci systému musí být použité stejné nebo kompatibilní technologie, které byly vybrány pro vývoj a správu aplikace Gloffer. Demonstrační aplikace musí být s aplikací Gloffer kompatibilní pro snadnou integraci řešení.
- Benevolentní licenční podmínky použitých aplikací třetích stran. Funkcionalitu a data poskytované těmito aplikacemi musí být možné využívat zdarma i pro komerční účely, tedy i pro e-commerce portál Gloffer zprostředkovávající prodej automobilů. Funkce musí být možné používat opakovaně bez vynucených pauz mezi voláními a bez limitu počtu volání v určitém časovém období. Aplikace nesmějí uživatele přesměrovávat mimo stránky webové aplikace.
- Rychlost odezvy vyhledávání odpovídající běžné míře podobných internetových webových aplikací.
- Uživatelská přívětivost obslužných prvků vyhledávání. Důraz není kladen na finální design a vzhled demonstrační aplikace, ale samotný způsob manipulace s formulářovými prvky vyhledání by měl odpovídat obecně očekávánému standardu.

5.3.3 Funkce demonstrační aplikace

Funkce systému jsou popsány tabulkou 5.1. Uživatelem je běžný uživatel s přístupem k demonstrační aplikaci.

Tabulka 5.1: Přehled implementovaných funkcí demonstrační aplikace

Událost	Reakce	Aktér
Určení výchozí lokace pomocí systému určování polohy	Nalezení (přibližné) lokace aktéra a nastavení lokace jako výchozí pro vyhledávání, zobrazení polohy na vložené mapě	Uživatel
Určení výchozí lokace manuálním výběrem z vložené mapy	Nastavení vybrané lokace jako výchozí pro vyhledávání, zobrazení polohy na vložené mapě	Uživatel
Určení výchozí lokace zadáním poštovní adresy	Nalezení odpovídajících souřadnic adresy a nastavení lokace jako výchozí pro vyhledávání, zobrazení polohy na vložené mapě	Uživatel
Vyhledání prodejců do určité silniční vzdálenosti (km)	Zobrazení lokací prodejců na mapě vložené do webové stránky; zobrazení adres prodejců seřazených dle silniční vzdálenost	Uživatel
Navržení trasy navštívení všech výsledků vyhledávání	Zobrazení nalezené okružní trasy na mapě vložené do webové stránky	Uživatel

5.4 Koncept

Návrh implementace vyhledávání se skládá z několika oblastí. Pro vyhledávání s využitím informací o poloze entit, musíme tyto lokační údaje nejdříve získat a registrovat v databázi. Následně ve webové demonstrační aplikaci vytvoříme uživatelské podpůrné prvky pro zadání výchozí zeměpisné polohy, vůči které se budou vyhledávat ostatní lokace dle vzdušné vzdálenosti. V závěrečné oblasti řešení se výsledky upřesní hledáním nejkratších cest pomocí podpůrné trasovací aplikace. Stejná aplikace je použita pro navržení efektivní dráhy mezi nalezenými lokacemi.

5.4.1 Oblasti řešení

Číselník adres a jejich zeměpisných souřadnic

Abychom mohli vyhledávat lokace prodejců registrovaných v aplikaci, musíme evidovat také jejich GPS souřadnice. Tyto lokační údaje můžeme požadovat od dotyčných subjektů při jejich registraci do systému. Ať už je osobou provádějící registraci laik (reprezentant prodejce) nebo pověřený a proškolený personál správce aplikace, vždy je zde riziko zadání nevalidních dat. Zadávající musí ověřit, že zadávané GPS souřadnice, má-li je vůbec k dispozici, skutečně odpovídají adrese registrovaného subjektu. Musí je rovněž zadat v korektním formátu v souřadném systému WGS84, se zeměpisnou šířkou a délkou v pořadí, v jakém jsou tyto hodnoty evidovány v databázi aplikace. Pokud zadávající uživatel nezná GPS souřadnice místa, musí je dohledat například použitím veřejně dostupných mapových aplikací. Podobné problémy mohou nastat při zadávání poštovní adresy, jejíž validita je také požadována, byť ne nutně pro vyhledávání dle lokačních údajů.

Řešením je vést v databázi číselník všech platných českých poštovních adres a jim odpovídajícím zeměpisným souřadnicím. Pro podporu registraci prodejce lze pak implementovat funkci asistenta pro vložení validní adresy, který uživateli nabízí varianty platných adres odpovídající právě zadávanému řetězci. Takový prvek se nazývá „našeptávač“[55]. GPS souřadnice pak uživatel nemusí zadávat vůbec, jelikož ty jsou pro adresy evidované v číselníku.

Snažíme se v co největší míře využít data uložená lokálně v databázi, protože každé volání API funkcí externích poskytovatelů by navyšovalo dobu vykonávání požadavku uživatele.

Prvním úkolem návrhu aplikace bude tedy navržení číselníku a jeho naplnění validními a kompletními daty.

Podpůrné prvky pro parametrizaci vyhledávání

Pro hledání lokací je potřeba zadat výchozí polohu. Typický dotaz vyhledávání bude: „najdi automobily v určité vzdálenosti od nějakého výchozího místa“, kde místo může být buď aktuální lokace uživatele, nebo adresa či GPS souřadnice zadané uživatelem při parametrizaci vyhledávání.

Dalším úkolem je tedy implementovat pomocné funkce usnadňující zadání této výchozí polohy:

- funkce pro získání aktuální polohy uživatele
- funkce pro zadání poštovní adresy pomocí našeptávače
- funkce pro zadání GPS souřadnic pomocí výběru z mapy vložené ve stránce aplikace

Pro uživatelovo vizuální ověření zadané polohy bude nutné zobrazit zvolené místo na mapě vložené ve webové stránce aplikace. Pro zobrazení vložené webové mapy byla vybrána aplikace Leaflet, viz kapitola 4.2, která splňuje nefunkční požadavky systému, viz kapitola 5.3.2.

Vyhledávání pomocí geolokačních dat

Hlavním cílem aplikace je poskytnout funkci vyhledávání pomocí geolokačních dat. Zadal-li uživatel výchozí lokaci a máme-li evidovány GPS souřadnice všech registrovaných prodejců, můžeme přistoupit k řešení samotného vyhledávání.

Na databázové úrovni jsme schopni vyhledávat pomocí vzdušné vzdálenosti mezi evidovanými body. Výsledek je třeba ještě upřesnit a přidat omezení, kde se zohlední volitelná maximální délka silniční trasy z výchozího bodu k nalezeným místům. Pokud je výsledků více, můžeme uživateli nabídnout možnost návrhu efektivní trasy pro navštívení všech nalezených míst. Pro tyto funkce vyžadující znalost topologie silniční sítě musíme využít rozhraní nějaké aplikace třetí strany, protože zamýšlený databázový model tyto prostorové informace neobsahuje.

Nastává zde však problém, jak takové API integrovat do parametrizovaného vyhledávaní. Zatímco u vyhledávání, kde se všechny atributy prodejců nacházejí v lokální databázi, lze parametry

triviálně zahrnout do dotazů a jejich výběr optimalizovat. V případě volání API externího poskytovatele služeb jsou možnosti optimalizace limitované. Už samotná výměna požadavku vyžaduje nějaký čas, který lze optimalizovat maximálně na straně infrastruktury naší aplikace. Úzkým hrdlem pak vždy zůstane server poskytovatele API a rychlost odezvy.

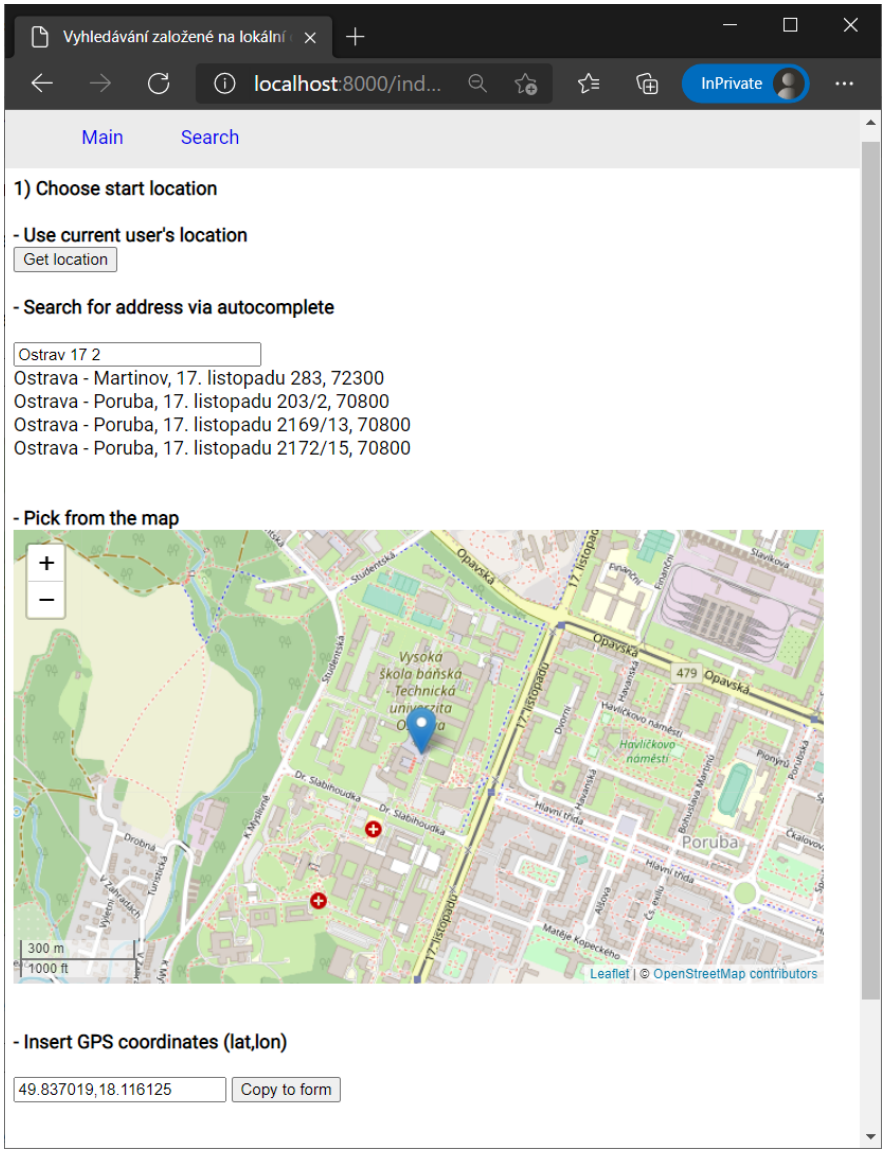
Zvolená pomocná aplikace Open Source Routing Machine (OSRM), viz kapitola 4.3, má tu výhodu, že může být hostovaná lokálně, tedy i na stejném serveru, na kterém je nasazena aplikace, která bude volat funkce tohoto rozhraní. Součástí lokálního nasazení mohou být všechny nutné mapové a trasové podklady z Open Street Map a OSRM aplikace pak nevyžaduje žádnou další externí komunikaci (může být používána zcela offline). OSRM splňuje i ostatní nefunkční požadavky systému, viz kapitola 5.3.2.

5.4.2 Použité technologie

- **MySQL**: aplikace je vyvíjena nad databázovým systémem MySQL, na kterém je částečně implementován i portál Gloffer.
- **PHP**: serverová část demonstrační webové aplikace je implementovaná v PHP, podobně jako systém Gloffer.
- **JavaScript**: pro některé podpůrné funkce uživatelského rozhraní (asistent pro zadání adresy), až po poměrně zásadní funkce aplikace (práce s vloženou mapou), je použit JavaScript.
- **AJAX**: pro funkci našeptávače, který asynchronně volá databázové dotazy hledající poštovní adresy, které vyhovují právě zadávanému textu.
- **Java**: syntaktická analýza číselníkových dat a jejich importování do databáze aplikace je implementována jako samostatná aplikace v jazyce Java.
- **XML**: výchozí formát vstupních dat z externího zdrojového číselníku.
- **Leaflet**: knihovna je použita pro vložení a manipulaci s webovou mapou.
- **Open Source Routing Machine (OSRM)**: tato aplikace se používá pro pokročilé trasovací funkce.
- **Docker**: virtualizační software pro spouštění kontejnerů představující nějaké aplikace, které pro svůj běh využívají funkce operačního systému, ve kterém je Docker spuštěn. Docker je použit pro lokální provoz aplikace OSRM.
- **JSON**: formát pro serializaci dat je v aplikaci využíván například při výměně dat mezi různými aplikačními vrstvami.
- **GeoJSON**: rozšíření formátu JSON je využito hlavně při prezentaci prostorových dat na vložené mapě.

5.4.3 Vzhled a aplikace

Aplikace má v prvé řadě demonstrovat použitelnost navrhovaného řešení. Na vzhled nebyl tedy kladen důraz a design aplikace je čistě funkční, viz náhled na obrázku 5.2.



Obrázek 5.2: Náhled na stránku demonstrační aplikace s vyhledávacím formulářem

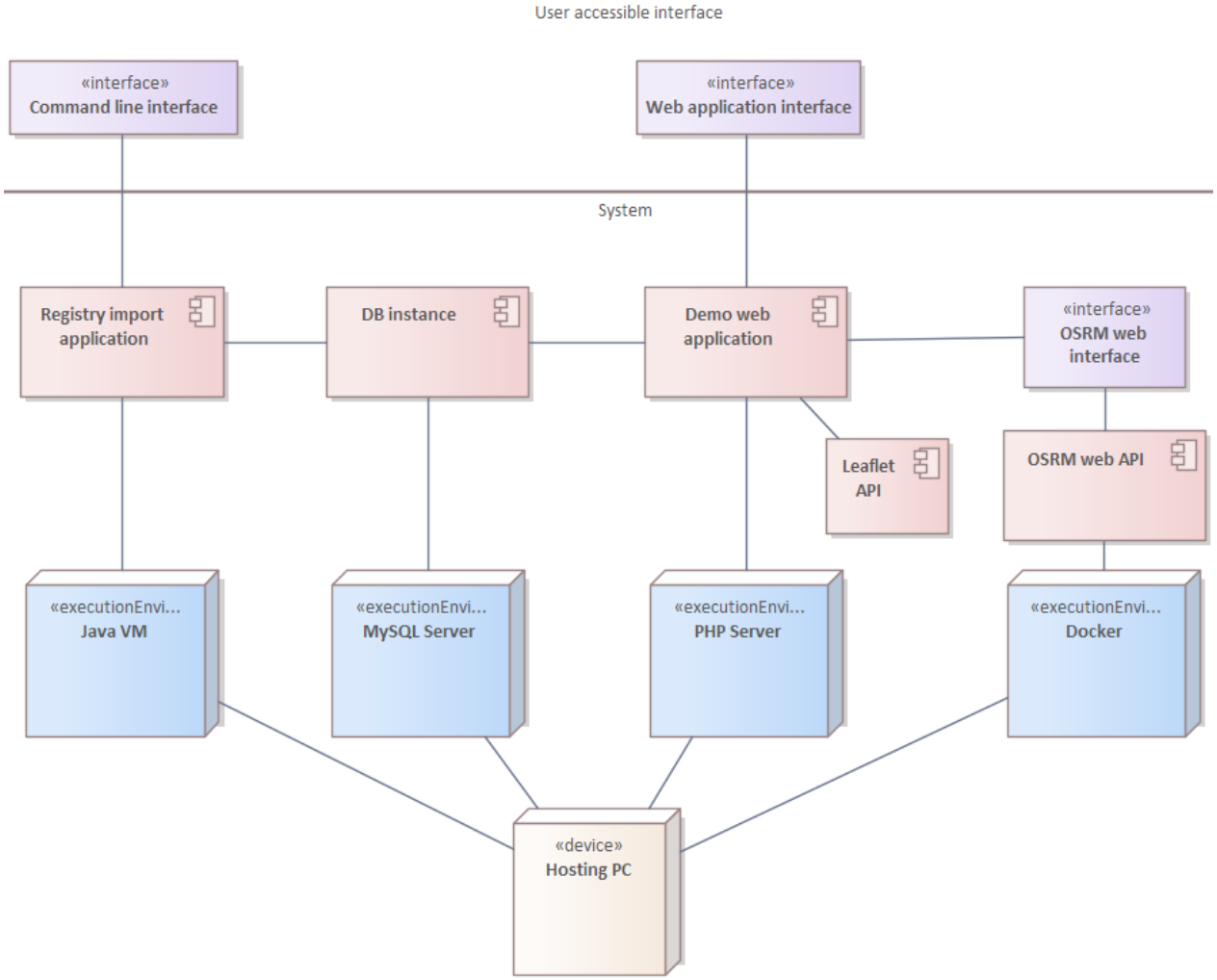
5.4.4 Architektura aplikace

Návrh řešení se skládá z dvou aplikací. První se zabývá čistě jednorázovým importem číselníkových dat, je implementována v jazyce JAVA a spouštěna z příkazové řádky. Data se ukládají do MySQL databáze, která musí mít již před spuštěním programu připravené navržené databázové schéma.

Očekává se, že pro použití aplikace je uživatel - administrátor obeznámen s principy jejího fungování, tedy s formátem dat na vstupu a cílovým databázovým modelem. V diagramu 5.3 je tato aplikace označena jako *Registry import application*.

Druhá část již představuje webovou aplikaci, která demonstruje použitelnost řešení pro koncového uživatele. Tato aplikace je implementována v PHP a funguje nad stejnou MySQL databází, ve které byl vytvořen číselník adres. V diagramu 5.3 je tato aplikace označena jako *Demo web application*.

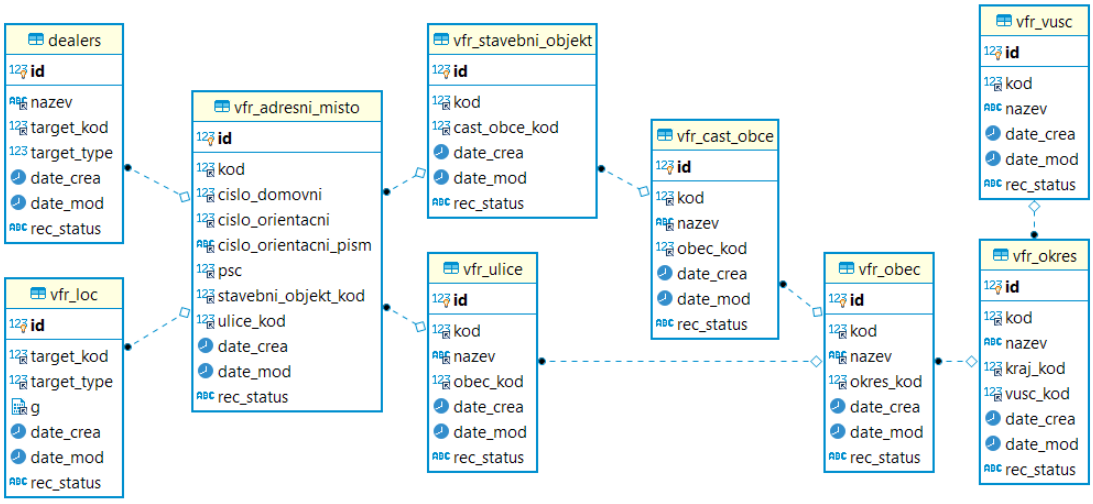
Webová aplikace používá aplikace a knihovny Leaflet a OSRM pro dodatečnou funkcionalitu. Leaflet je importována do webové aplikace jako JavaScript knihovna. Aplikace OSRM je nasazená ve virtualizační platformě Docker, dosažitelná jako lokální webový server a její rozhraní je přístupné pomocí HTTP protokolu.



Obrázek 5.3: Architektura navrženého řešení

5.4.5 ER diagram

Databázové schéma demonstrační aplikace se skládá z číselníkových tabulek (jejich názvy mají prefix VFR_) a z tabulky DEALERS pro evidenci prodejců. Číselníková data se budou měnit jen při plánovaných aktualizacích číselníku, jinak budou uživatelé a procesy vykonávat pouze SELECT operace. Nemusíme tedy řešit dopad existence většího množství indexů na UPDATE a INSERT operace.



Obrázek 5.4: ER diagram aplikace

Kapitola 6

Implementace

Cílem kapitoly je představit poznatky získané během implementace demonstrační aplikace. Kapitola je členěna vzhledem k oblastem řešení. Text se soustředí na detaily specifické pro zadání práce a nezabývá se tedy oblastmi, které přímo nesouvisí se vstupními požadavky (např. architektura demonstrační webové PHP aplikace).

6.1 Implementace vytvoření číselníku adres a souřadnic

Aplikace pro import číselníkových dat je napsána v jazyce Java a není součástí hlavní demonstrační webové aplikace, ve kterém je realizováno uživatelské GUI a funkce vyhledávání. Nicméně obě aplikace pracují nad stejným databázovým schématem. Číselníková aplikace nemá žádné grafické uživatelské rozhraní a je spouštěna z příkazové řádky. Jejím jediným účelem je jednorázový import číselníkových dat.

Jako zdroj dat byl vybrán registr RÚIAN, který byl představen v kapitole 2.2.1, včetně důvodů objasňujících jeho volbu.

Aplikace nemá rozšířené možnosti parametrizovaného vstupu a očekává přítomnost vstupních XML souborů získaných ze zdrojového systému (VDP RÚIAN) v předem definovaném adresáři. Po spuštění pak aplikace prochází každý ze vstupních souborů zvlášť a nad každým provede následující kroky:

1. Proces provede syntaktickou analýzu (parsing), která vrací objekt `VDPData`.
   (a) Ten obsahuje kolekce POJO[1] objektů (např. `Obec, AdresniMisto`), které v míře reflektující požadavky projektu odpovídají vybraným prvkům z datového modelu RÚIAN.
2. Ještě během parsingu se provádí transformace všech načtených souřadnic z S-JTSK do WGS84.
3. Vrácená data z jednoho načteného XML se předají třídě, která je uloží do databáze.

[1] Plain Old Java Object – jednoduchý objekt nezatížený restrikcemi.

Po posledním kroku se načte další XML soubor ve vstupním adresáři a proces se opakuje.

6.1.1 Syntaktická analýza vstupních XML

Pro implementaci parseru (programu vykonávající syntaktickou analýzu textu) byla zvolena knihovna StAX, která je součástí standardního JAVA XML API.

StAX pracuje ze zdrojovým XML jako s „proudem dat“ (dále již „stream“). Jeho API je založeno na principu vzoru Iterátor. Voláním metody `next` iterátor postupně prochází streamem, vždy položku po položce, kde položkou může být počáteční element (start tag), textový prvek (text node) nebo komentář. Iterátor po načtení elementu poskytuje pomocí getters metod mimo jiné přístup k atributům elementu.

Jiný často používaný přístup parsování XML spočívá v tom, že se obsah celého zdrojového XML transformuje do objektů, jejichž třídy odpovídají modelu XML. Obsah zdrojového XML je tak k dispozici v paměti a je možno se všemi informacemi dále libovolném pořadí pracovat, nehledě na aktuální pozici iterátoru, jako v případě postupného načítání streamu.

Vzhledem k tomu, že zdrojová XML obsahují velké množství informací, které pro lokální číselník nepotřebujeme, streamové zpracování umožňuje snadno ignorovat obsah elementu, který nás nezajímá. Cílem importu je data načíst a uložit je do databáze. Jelikož nad nimi nepotřebujeme provádět při importu žádné složitější operace (pouze konverzi mezi datovými formáty), nepotřebujeme je ani držet v paměti ve složité stromové struktuře dané XML formátem.

Elementy a atributy, které jsou pro aplikační číselník podstatné, jsou uvedeny v kapitole 2.2.1. Návrh tříd odpovídá těmto vybraným entitám, kde jednotlivé třídy odpovídají vybraným elementům ve zdrojových XML souborech.

Načtení vstupních souborů

Aplikace se spouští z `main` metody třídy `ImportRegistryData`. Ta v cyklu načítá obsah předdefinovaných adresářů, jejichž názvy jsou definované v konstantách `FS_VDPFolder_Stat` a `FS_VDPFolder_Obec`, a nad každým načteným souborem volá metodu `parseFile`. Vstupní adresáře jsou dva, jelikož RÚIAN soubory pro detaily obcí se liší od souboru s detaily státu, přičemž platí, že oba typy souborů obsahují data, která musíme evidovat.

Analýza souborů

Analýzu souborů obstarává třída `StAXParser`, volaná z obslužného objektu `VDPParser`, která prochází načtený XML soubor jako stream a kontroluje názvy načtených počátečních elementů, zda odpovídají objektům, které je třeba evidovat v číselníku. Na základě různých typů entit probíhá pak uložení do odpovídajícího POJO objektu.

Všechny třídy reprezentující entity číselníků RÚIAN jsou potomky třídy `VFRData`. Předpokládáme, že zdrojový číselník obsahuje jen konzistentní a validní data, proto při importu neprobíhají

žádné validace načtených hodnot. Pro zjednodušení procesů syntaktické analýzy a následného uložení dat se v třídním modelu nijak nereflektují vazby mezi entitami (např. *ulice leží v obci*). Tato provázanost je realizována pomocí identifikátorů, vlastních i cizích, v jednotlivých entitách zdrojového registru. Stejné identifikátory jsou transformovány do interního databázového číselníku a konzistence dat je ověřena teprve po aktivaci definovaných integritních omezení na konci celého procesu importu dat. Proto parsovací metoda po dokončení analýzy každého souboru vrací objekt `VDPData`, který obsahuje kolekce načtených entitních objektů, nehledě na hierarchickou strukturu dat.

Transformace souřadnic

Entity, které obsahují GML element, tedy v případě dat z RÚIAN zeměpisné souřadnice v souřadném referenčním systému S-JTSK, volají `setGmlPos` metodu. Ta je definovaná ve společné třídě `VFRData`, která je nadřazená ostatním třídám reprezentující RÚIAN entity. `SetGmlPos`, kromě uložení načtené a nezměněné hodnoty ze vstupního XML, volá statickou transformační funkci, která převede souřadnice z S-JTSK systému do systému WGS84 (GPS). Získané souřadnice (zeměpisná šířka a délka) jsou uloženy do atributu `GeoCoord` nadřazeného objektu `VFRData`. Pro transformaci byl použit algoritmus [56].

6.1.2 Dávkové uložení dat do DB

Po každém analyzovaném a načteném obsahu jednoho dávkového XML se provede uložení dat do lokální databáze.

Před započetím celého procesu je však nutné databázi připravit pro hromadný import dočasnou deaktivací indexů a integritních omezení.

Proces pro ukládání dat je implementován s využitím návrhových vzorů Data Access Object (DAO) a Data Transfer Object (DTO) s maximálním využitím dávkových databázových operací pomocí JDBC objektů `PreparedStatement`. Po uložení obsahu jedné vstupní dávky se provede potvrzení transakce a řízení programu pokračuje od začátku načtením dalšího vstupního XML.

Po skončení celého importu je nutné v DB znova aktivovat indexy a integritní omezení. Zvláště vytvoření prostorového indexu R-Tree může být nad velkým množstvím uložených zeměpisných souřadnic časově velice náročné. Vytvoření indexu je však jednorázová akce, která se provede pouze při prvotním vytvoření číselníku, do kterého se pak již data nepřidávají.

Vstupní objekt

Po dokončení syntaktické analýzy všech vstupních souborů jsou všechna načtená data k dispozici v objektu `VDPData`, který je atributem obslužné třídy `VDPParser`, kterou byla spuštěna analýza načteného souborů. Načtená data jsou tedy uložena v paměti a zbývá uložit je do databáze.

Příprava dat

Hlavní řídící třída pro vložení dat do databáze je `Queries`. Její metoda `storeVDP(VDPData data)` je volána po analýze jednoho celého vstupního souboru. Metoda naváže spojení s databází pomocí třídy `ConnectionPool` a pak prochází všechny atributy vstupního objektu `VDPData`, jimiž jsou kolekce načtených entit z registru RÚIAN. Pro každou entitu je v `Queries` definována specifická metoda s názvem `qCreate{název entity}All` (např. `qCreateObecAll`). Metoda inicializuje objekt implementující konkrétní DAO rozhraní pro onu entitu a provede transformaci kolekce entit do kolekce DTO objektů.

Vložení dat

Kolekce DTO objektů se pak předá konkrétní implementaci příslušné DAO metodě pro vložení dat do databáze. Pro komunikaci s databází jsou používány `PreparedStatement` objekty a metody. Definuje se řetězec představující příkaz INSERT, kde parametry jsou nahrazeny zástupnými symboly otazníků. Pomocí `PreparedStatement.set*` metod se v cyklu nastaví parametry příkazu na konkrétní hodnoty prvků z kolekce DTO objektů a každý takto připravený příkaz se uloží do dávky pomocí metody `PreparedStatement.addBatch`. Po připravení celé dávky se data vloží do databáze pomocí příkazu `PreparedStatement.executeBatch`.

Tímto způsobem se vloží data všech entit načtené z jednoho XML souboru z RÚIAN. Pokud během vložení nebyla zachycena žádná chyba, provede se potvrzení transakce.

6.1.3 Návrh schématu s přihlédnutím k prostorovým funkcím v MySQL

Souřadnice evidujeme v souřadném systému WGS84, jelikož podpora GPS zařízení a aplikací je jedním z požadavků demonstrační aplikace. Tato data ukládáme do tabulky `VFR_LOC`. Ve verzích MySQL s podporou novějšího formátu úložiště dat InnoDB je možné nad prostorovými daty vytvořit prostorový index typu R-Tree. Aby bylo možné sloupec indexovat, je nutné splnit tyto podmínky:

1. sloupec musí být definován jako prostorový datový typ (např. `geometry`),
2. sloupec musí být definován jako NOT NULL,
3. sloupec musí mít definován SRID identifikátor.

Kvůli druhé podmínce jsou souřadnicová data oddělena od ostatních atributů adresy pro případ, kdy k adrese nebyly souřadnice ještě vloženy.

Dle třetí podmínky je sloupci nastaveno SRID na hodnotu 4326, pro systém WGS84.

Pouze po těchto podmínkách lze prostorová data indexovat. Podpora indexu i tak není v případě MySQL 8.0 u mnoha prostorových funkcí samozřejmá. Tomuto problému jsme se již věnovali v kapitole 3.8.1 na stránce 32.

Ukázka syntaxe pro definici sloupce a indexu je prezentována na výpise 6.1.

```
g GEOMETRY NOT NULL SRID 4326;
CREATE SPATIAL INDEX `g` ON `vfr_loc` (`g`) VISIBLE;
```

Listing 6.1: Vytvoření sloupce a indexu nad prostorovým datovým typem

6.2 Implementace funkcí pro zadání lokace

Nyní se v popisu implementace dostáváme k webové demonstrační aplikaci, která má za cíl prezentovat uživatelskou stránku navrhovaného řešení. Aplikace je napsána v jazyce PHP a pracuje nad stejným databázovým schématem, ve kterém byl vytvořen lokální číselník poštovních adres a GPS souřadnic.

Funkcemi pro zadání lokace se myslí funkce, které může uživatel použít pro definici výchozí polohy, vůči které bude iniciovat vyhledávání ostatních prostorových dat. Uživatel má tyto možnosti:

- vybrat polohu z vložené mapy,
- zadat poštovní adresu (s možným využitím našeptávače),
- povolit automatizované zjištění své aktuální polohy,
- zadat GPS souřadnice ručně do textového pole formuláře.

Pro výběr adresy z mapy použijeme knihovnu Leaflet. Do stránky bude vloženo okno s interaktivní webovou mapou. Kliknutím do ní bude možné zvolit souřadnice.

Zadání poštovní adresy uživateli ulehčíme pomocí prvku našeptávač. Během zadání adresy se budou uživateli nabízet nalezené adresy, ze kterých může provést výběr ještě před dokončením ručního zadání zamýšlené adresy.

Pro automatizované zjištění polohy použijeme funkci HTML5 `getCurrentPosition`.

Poslední možností, ručním zadání souřadnic, se nebudeme blíže zabývat. Zodpovědnost za správné vložení dat je na uživateli. Nicméně nehledě na zvolenou možnost, vybraná poloha bude vždy zobrazena na vložené webové mapě. V případě kompletního ručního zadání souřadnic bude aplikace uživateli nápomocná alespoň tak, že bude moci vložené mapy vizuálně ověřit správnost zadání dat. V případě nutnosti pak může uživatel zadání opravit.

6.2.1 Zadání pomocí vložené mapy

Pro umožnění zadání polohy výběrem z mapy je využita knihovna Leaflet.

Jako zdroj mapových podkladů je nastavena aplikace Open Street Map. Nad objektem mapy je definována událost kliknutí myší. Ta vrací GPS souřadnice odpovídající místu kurzoru nad mapou při kliknutí. Souřadnice se zobrazí ve vstupním formuláři pro parametrizaci vyhledávání. Zároveň se v mapě vytvoří značka ukazující vybranou polohu.

Inicializace mapy

Funkce pro práci s knihovnou Leaflet jsou definovány v souboru `scripts.js`. Mapa je inicializována voláním funkce `embeddedMap` z PHP skriptu při vykreslení stránky s vyhledávacím formulářem. V `emdeddedMap` funkci se provede inicializace mapy voláním metody `L.map`, (objekt `L` je automaticky inicializovaný při importu Leaflet knihovny). Metoda vrátí instanci mapy, se kterou dále pracují metody definované ve `scripts.js`. Provede se vycentrování mapy na určitou polohu s určitou mírou přiblížení pomocí funkce `centerMap`. Toto nastavení je uloženo v konstantách skriptu a bylo by možné je externalizovat do nějakého sdíleného registru konfigurace, nicméně vzhledem k účelům demonstrační aplikace toto nebylo realizováno. Posledním krokem inicializace je nastavení události `click`, vyvolávající funkci `onMapClick`.

Událost kliknutí

Při kliknutí do mapy je knihovnou Leaflet do funkce `onMapClick` předán parametr `e` (event), ze kterého lze získat souřadnice zvoleného místa (`e.latlng.lat, e.latlng.lng`).

Souřadnice se předají do funkce `updateStartingLocation`, která na mapě zobrazí vybranou lokaci a aktualizuje vyhledávací formulář o souřadnice zvolené polohy. Tato funkce je volána i v jiných případech, kdy je potřeba na mapě zobrazit uživatelem zvolenou výchozí polohu, ne jen pro událost kliknutí.

6.2.2 Vložení adresy pomocí našeptávače

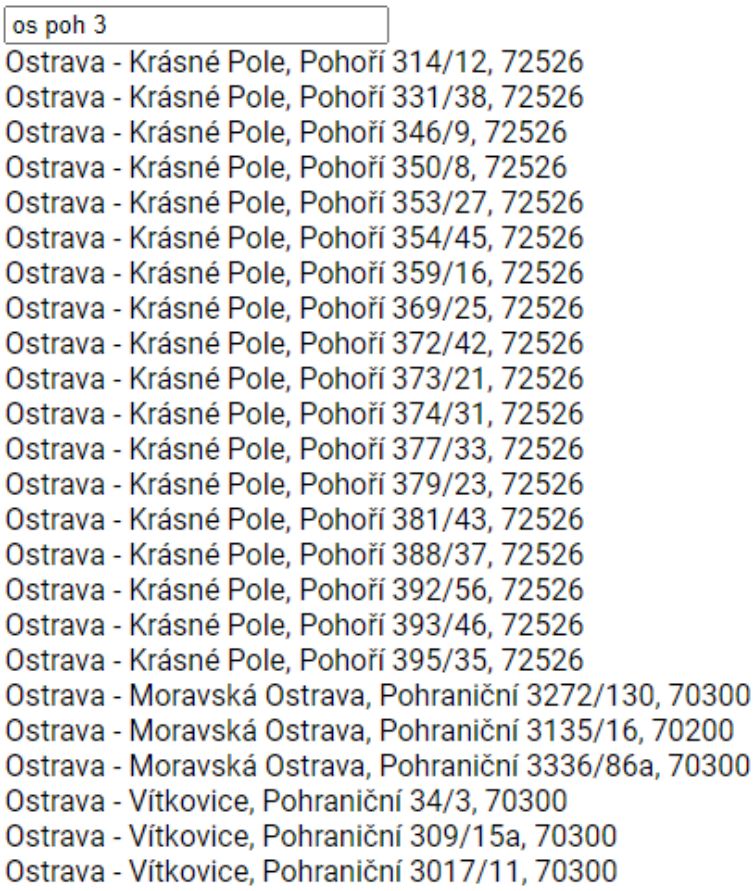
Pro manuální zadání adresy slouží formulářové textové pole, jehož obsah se však zpracovává již během samotného vkládání textu. Uživatel tedy dostane od systému zpětnou odezvu ještě předtím, než formulář po kompletním dokončení zadání odešle. Může si tedy vybrat i z většího počtu nalezených adres, aniž by musel zadat adresu kompletní. To jednak usnadní zadání adresy uživateli známé, ale pomůže i v případě nejasností u některého z prvků adresy. Pokud si uživatel například přesně nevybavuje číslo popisné, může si vzpomenout ve chvíli, kdy mu bude nabídnut seznam adres, které vyhovují dosavadnímu vstupu.

Funkce je implementovaná pomocí knihoven jazyka JavaScript pro asynchronní procesy (AJAX). Asynchronně se volá URL adresa, která je poskytována PHP serverovou implementací aplikace. Volaná PHP funkce spouští vyhledávání možných adres nad databází a vrací výsledky zpět.

Způsoby zadání a zpracování textu fungují na stejném principu, jako např. vyhledávání v jízdních řádech aplikace idos.cz. Stačí tedy zadat jen libovolný počet počátečních písmen slov oddělené mezerou. Očekává se vstup ve formátu v pořadí obec, ulice a domovní[2] číslo nebo též obec, část obce a domovní číslo pro adresní místa bez určené ulice.

[2]domovním číslem se rozumí číslo popisné (unikátní v rámci jedné obce), nebo číslo evidenční (stavby dočasné, pro rodinnou rekreaci atd.). Každá adresa v ČR má mít právě jedno z těchto dvou čísel, které se obě nazývají číslem domovním.

Příklad: vstup „os poh 3“ nabídne adresy s domovním číslem začínajícím 3 v ostravských ulicích Pohoří (městská část Krásné Pole) a Pohraniční (městské části Vítkovice, Moravská Ostrava), viz obrázek 6.1. Formulář aktualizuje a vrací výsledek po splnění několika podmínek:



Obrázek 6.1: Použití formulářového našeptávače

1. Jsou zadané minimálně dva znaky oddělené mezerou: tím je zajištěno, že se výsledky omezí nejen na obce ale i na ulice. Bez tohoto omezení by bylo výsledků příliš, což by znemožňovalo nad nimi pohodlně provádět výběry.

2. Uplynulo 500ms od posledního vloženého znaku: tím je zajištěno, že se nedotazuje databáze po každém vstupu, i když uživatel chce napsat znaků více a upřesnit tak zadání vyhledávání. Pouze po této krátké prodlevě se spustí hledání.

3. Celý vložený řetězec se liší od předešlého: tím se ignorují údery klávesnice, které vstup nemění, jako např. použití kurzorových šipek.

Databázový dotaz pak hledá záznamy, které začínají stejně (nemusí jít o shodu celého řetězce) jako vstupní parametry. Počet výsledků je omezen konfigurovatelným parametrem. Uživateli se

tedy vrací seznam nalezených adres, ale pokud bylo výsledků příliš mnoho, tedy více, než je limit definovaný parametrem, uživatel je na toto větší množství upozorněn a může svůj dotaz upřesnit.

Data (výsledky) se z asynchronního volání zobrazují v předem umístěných elementech stránky pomocí JavaScriptu, kam se z PHP předávají ve formátu JSON. Uživatel provede výběr výsledků kliknutím na jeden ze záznamů.

Formulářová komponenta

Formulářová komponenta realizující dynamické a asynchronní vyhledávání adres je definována jako PHP funkce `cmp_autocompleteAddress`. Po jejím volání se do stránky vloží připravené formulářové prvky. Jmenujme hlavně vstupní textové pole s definovanou HTML událostí `onkeyup`, která v případě umístění textového kurzoru do tohoto prvku poté při každém stisknutí klávesy volá JavaScript funkci `findSimilarAddress` (ze souboru `scripts.js`). Dále bude formulář obsahovat blokový HTML prvek pro zobrazení nalezených mezivýsledků. Identifikátory těchto prvků jsou konfigurovatelné v parametrech komponentní funkce, nicméně parametry mají i definované výchozí hodnoty, které je doporučeno použít.

Příprava asynchronního volání

Pro zamezení neefektivního a zbytečného vykonávání asynchronních volání je nejdříve provedena sada kontrol pomocí JavaScript funkce `findSimilarAddress`. Té je parametry předáván vložený vstupní text `inputStr` a názvy identifikátorů HTML elementů pro následné zobrazení mezivýsledků jako součástí formulářových polí.

Porovná se, že zadaný text `inputStr` se liší od textu, který byl zadaný naposledy a který je definován jako globální proměnná `lastStr`. Tím se zamezí spouštění dalšího hledání adresy pro stejný vstup, což by jinak nastalo v případě stisku kláves, které vstupní text nemění, ale vyvolávají událost `onkeyup`.

Dále se vstupní text rozdělí na parametry dle mezer a pokud je počet parametrů zadané adresy nedostatečný, dále se nepokračuje.

Poslední kontrolou je ověření prodlevy od posledního volání asynchronní funkce. Tato doba je definovaná jako konfigurovatelný parametr `$config['JSParams']['doneTypingInterval']` a cílem kontroly je, aby funkce nebyla volána příliš často a ještě před tím, než uživatel přestal zadávat text v očekávání obdržení odezvy asynchronního volání funkce. Toto je realizováno pomocí JavaScript funkce `setTimeout`, které se parametrem předá `findSimilarAddressNow` jako callback funkce společně s parametrem doby prodlevy. Funkce `findSimilarAddressNow` již provádí samotné asynchronní dotazování a vrácení výsledků.

Asynchronní volání

Funkce nejdříve definuje callback funkci pro zachycení výsledku následného asynchronního volání. Blok kódu, který zpracovává asynchronně získaný výsledek, je předán JavaScript funkci `xmlhttp.onreadystatechange`. V případě, že server odeslal odpověď se stavy „request finished and response ready“ a „OK“ (`readyState == 4 && status == 200`), proběhne zpracování výsledku.

Po definici callback funkce následuje asynchronní volání `findSimilarAddress.php?place`, kde parametr `place` odpovídá textu zadanému uživatelem. Po odeslání tohoto požadavku funkce končí a další kroky budou vykonávány, jakmile definovaná callback funkce obdrží serverovou odpověď.

Hledání adres dle vstupního řetězce

Hledání podobných adres je implementováno v PHP s využitím MySQL databáze. Skript má povinný parametr `place`, předaný v GET požadavku, který představuje text zadaný uživatelem. Ten se rozdělí do pole dle mezer pomocí funkce `explode` a pomocí těchto částí adresy se vytvoří objekt `Adresa`. Implementované databázové funkce používají objekty třídy `Adresa` právě pro předání vstupních a výstupních hodnot. Jednou z těchto DB funkcí je i `findSimilarAddress` pro nalezení adres, které alespoň částečně odpovídají vstupním parametrům.

Dotaz vyhledávání je triviální. Z databázového pohledu `v_vfr_adresa`, který představuje seznam všech poštovních adres číselníku, se pomocí LIKE operátoru hledají záznamy, jejichž prefixy obcí, ulic a domovních čísel se shodují s hodnotami v parametrech funkce. Tento dotaz může použít indexy nad zmíněnými sloupci, protože parametry operátoru LIKE začínají konkrétními hodnotami. Pokud bychom hledali jen pomocí sufixů a začátek textů by mohl být libovolný, index by nebyl použit.

Získaná data jsou transformována do DTO objektu `FoundAddressesDTO`, který obsahuje kolekci adres a příznak s informací, zda bylo možné získat více adres nad limit daný konfigurovatelným parametrem (`$config['DBParams']['findAddressLimit']`). Na závěr se objekt převede formátu JSON a vrátí se, což vyvolá událost, která spustí zpracování tohoto výsledku pomocí JavaScriptu.

Zpracování výsledku asynchronního volání

Výsledky funkce hledající poštovní adresy jsou získány jako JSON objekt, obsahující kolekci nalezených adres. Pro deserializaci vstupu na objekty JavaScript se použije funkce `JSON.parse`. Následně se prochází kolekce objektů a do řetězce `foundResultsNew` se vkládají HTML elementy pro prezentaci těchto výsledků. Na konci zpracování se tyto elementy vloží do HTML elementu opět pomocí funkce přistupující k DOM stránky.

Výsledky hledání vložených pomocí JavaScriptu dynamicky do těla stránky jsou rozšířeny o funkci, která v případě, že je na jeden z výsledků kliknuto, uloží vybranou adresu do vstupního formulářového pole.

Možnosti budoucího rozšíření

Hlavní nevýhodou prvku našeptávače pro manuální vložení adresy, je zvolený způsob postupného zadávání adresy. Ten v případě mnohoslovných jmen obcí a ulic neumožňuje zadat přesnější informaci s cílem snížit počet kandidátů ve výsledku. Prvek totiž vždy očekává právě trojici údajů: „obec, ulice (nebo část obce) a domovní číslo“.

Jiný způsob řešení by mohl být ten, kdy by uživatel postupně vyplňoval části adresy, od větších celků po menší. Podobnou realizaci lze odzkoušet např. na sreality.cz. I to má však svá úskalí. Pokud uživatel zná adresu, kterou zadává, pak tento přístup je těžkopádnější. Musí se nejdříve napsat a zvolit obec a až poté zadat ulice. Pokud by se zadala nejdříve ulice, filtr by musel vybrat všechny ulice se stejným názvem, kterých může být velké množství. Filtr na sreality.cz logicky nabízí nejdříve ulice z větších měst, což nemusí být vždy pro uživatele žádoucí. Dá se říct, že tento přístup je vhodnější pro případ užití, kdy nevíme přesně, jakou adresu hledáme a postupným zadáváním územních celků se pokračuje hlouběji ke konkrétnímu výběru. Kdežto účel implementované funkce je umožnit co nejrychlejší zadání adresy dle validního číselníkového údaje. Zvolený způsob se proto jeví vhodnějším.

6.2.3 Automatizované zjištění lokace

Pro automatizované zjištění polohy, které je zároveň uživatelsky velmi komfortní, lze využít HTML5 Geolocation API a volat funkci `navigator.geolocation.getCurrentPosition()`. API je široce podporováno napříč prohlížeči. Před každým prvním voláním funkce je uživatel vyzván internetovým prohlížečem pro udělení souhlasů k získání polohy.

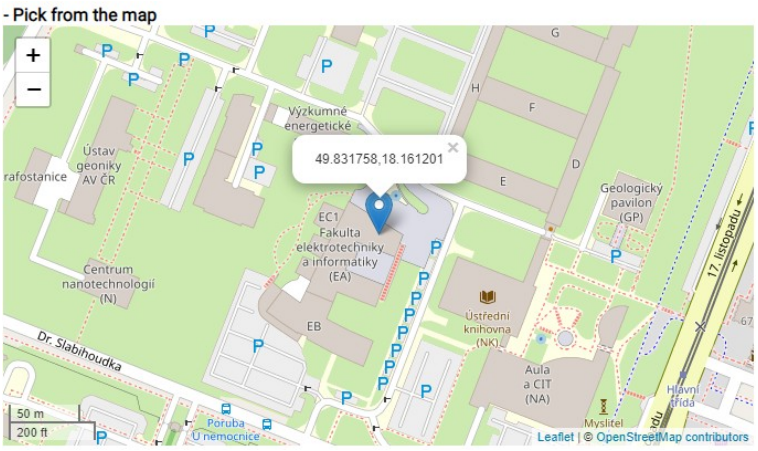
Funkce je zcela závislá na informaci o lokaci, kterou poskytuje uživatelovo zařízení samotné. Zařízení určuje lokaci typicky pomocí GPS modulu, je-li tento přítomen, nebo odvozuje lokaci z uživatelovy IP adresy, RFID, WiFi a Bluetooth MAC adres, či dle ID GSM/CDMA [57].

Funkce rozhraní podporují pouze souřadný systém WGS84.

Tato funkce však nemůže garantovat přesnost nalezené lokace. Uživatel by proto měl mít možnost volbu ručně opravit, k čemuž může využít ostatní představené možnosti zadání polohy. Každopádně bude nalezená lokace zobrazena ve vložené webové mapě, která bude na nalezenou polohu vycentrována.

6.2.4 Zobrazení lokace na vložené mapě

I v případě, že uživatel zvolí polohu bez použití vložené mapy, tedy pomocí automatizovaného zjištění polohy, zadání adresy či zadání souřadnic, zobrazí se zvolená poloha v mapovém objektu. Pro tyto účely se volá funkce `updateStartingLocation`, které se souřadnice předají jako parametr. Uživatel tak získá vizuální zpětnou vazbu o zadané poloze a může z mapového zobrazení (ukázka na obrázku 6.2) vyvodit, zda je lokace zadaná správně, či je nutné ji opravit. To může provést přímo kliknutím do mapy a může tedy metody zadání lokace takto kombinovat.



Obrázek 6.2: Zobrazení výchozí lokace na mapovém prvku Leaflet

Zobrazení zvolené polohy na mapě

Funkce `updateStartingLocation`, které jsou souřadnice předány v parametru, provede následující kroky:

1. Na zvolených souřadnicích je v mapě vytvořena značka pomocí metody `L.marker`, které se předají zvolená souřadnice a text značky (tvořen v tomto případě souřadnicemi samotnými, ale tento text je konfigurovatelný). Před vytvořením značky se odstraní předchozí značky, aby se mapa při každém kliknutí neplnila dodatečnými grafickými prvky.

2. Mapa je na toto místo vycentrována.

3. Pomocí funkce `document.getElementById` z rozhraní DOM se zvolené souřadnice vrací ze skriptu do stránky s formulářem a nastaví se jako atribut `value` formulářového vstupního pole s identifikátorem `startingLocation`. Vybraná poloha je tak nastavena jako výchozí poloha pro další vyhledávání.

6.3 Implementace vyhledávání pomocí geolokačních dat

Uživatel tedy již zadal lokaci, vůči které chce spustit vyhledávání ostatních poloh (adres prodejců). V případě začlenění tohoto vyhledávání do konkrétní e-shop aplikace, by formulář obsahoval i parametry vyhledávaného zboží, jako např. cenu, typ produktu, barvu, stáří a další. Tyto predikáty by při vykonání vyhledávacího dotazu omezily výslednou množinu dat, nicméně pro demonstraci vyhledávání prostorových dat nejsou zapotřebí a nebudou tedy v popisu implementace (a ani v implementaci samotné) reflektovány.

Jediným dalším parametrem, který musí uživatel zadat a který je klíčový pro tuto práci, je maximální délka silniční trasy d nutná pro dosažení vyhledávaných prodejců.

Kroky implementace jsou následující:

1. Na databázové úrovni vyhledáme lokace, které jsou v dosahu vzhledem k vzdušné vzdálenosti.
2. Upřesníme výsledky omezením filtru na délku silniční trasy.
3. Zobrazíme finální výsledky hledání v přehledové tabulce a také na mapě vložené do stránky.
4. Zobrazíme efektivní trasu pro navštívení nalezených lokací.

Na databázové úrovni máme k dispozici data číselníku všech poštovních adres ČR a jejich GPS souřadnic a také detaily prodejců a jejich vazeb (n:1) na tabulku číselníku. Nemáme však informace o silniční síti, např. v podobě prostorových dat typu (Multi)LineString (linie). Můžeme tedy pouze porovnávat výchozí lokaci s lokacemi prodejců a měřit jejich vzájemnou vzdušnou vzdálenost. Ta bude vždy menší nebo rovna délce silniční trasy a proto omezením dotazu na vzdušnou vzdálenost nehrozí riziko opomenutí dat ve výsledné sadě.

Pomocí databázového dotazu tedy omezíme množinu vhodných kandidátů dle skutečných silničních vzdáleností, které získáme z aplikace OSRM, viz kapitola 4.3. Počet nutných ověření záleží na velikosti první sady výsledků, a tedy na prvotních parametrech vyhledávání. Ale pokud tyto parametry nebyly nastaveny příliš velkoryse, bude v této sadě kandidátů nepoměrně méně, než kdyby se pomocná aplikace volala pro každého prodejce zaregistrovaného v databázi.

Zjištění efektivní trasy pro navštívení všech lokací je realizováno opět pomocí OSRM.

6.3.1 Vyhledání dle vzdušné vzdálenosti

Naivním způsobem vyhledávání by bylo porovnat vzdálenosti mezi výchozím bodem a lokacemi všech prodejců a vrátit jen ty výsledky s hodnotou vzdálenosti $<= d$. Takový dotaz by vedl k průchodů všech řádků s lokacemi prodejců a byl by použitelný jen v případě malého celkového počtu záznamů.

Množinu porovnávaných bodů musíme tedy nějak efektivně a validně redukovat.

Hledáme polohy prodejců, které leží v kruhu, jehož střed je uživatelem zadaná lokace a poloměr je zadaná vzdálenost d. Nabízí se použít MySQL funkci `ST_Buffer(střed, poloměr)`, která takovýto kruh dokáže definovat. Bohužel i v aktuální, ke dni 31.3.2021, verzi 8.0.23 (poslední oficiální stabilní verze z 18.1.2021) však MySQL v této funkci podporuje pouze souřadný systém SRID = 0 a není tedy vhodná pro indexované GPS souřadnice. Index by zcela ignorovala. Nicméně se zdá, že toto bude brzo napraveno. Oficiální MySQL dokumentace již zmiňuje podporu geografických souřadných systému i ve funkci `ST_Buffer`, která by měla být dostupná ve verzi 8.0.25. Souhrnný popis ani datum dostupnosti této verze nejsou ke dni 31.3.2021 známy.

Dodejme, že v případě databázového systému PostGIS je implementace funkce `ST_Buffer` vyhovující, jelikož podporuje geografické souřadnicové systémy a dokáže využít prostorového indexu, je-li nad daty definován.

Další variantou by mohla být funkce `ST_Envelope` pro vytvoření obálky dle zadaných bodů mnohoúhelníku. Bohužel ani tato funkce nepodporuje geografické souřadné systémy.

Nezbývá tedy než oblast kandidátů definovat ručně pomocí definice ohraničujícího čtverce, jehož body jsme schopni relativně snadno odvodit. Pokud pak porovnáváme prostorové body ohraničené takovýmto objektem, MySQL je schopen použít prostorový index pro omezení všech registrovaných lokací jen na ty, které leží uvnitř či na hraně této oblasti. Tím dramaticky snížíme počet řádků, pro které je nutné porovnat jejich vzdálenost ke středovému bodu.

Příprava minimálního ohraničujícího obdélníku

Pro vytvoření čtvercové obálky musíme zadat souřadnice jejich dvou protilehlých bodů. Musíme vypočítat posun výchozího středového bodu o $\pm d$. Pro koordináty v souřadném systému WGS84 byl použit algoritmus [58]. Vypočítané hodnoty posunu přičítáme a odečítáme vzhledem ke středovému bodu. Získáme tak 4 rohové body (A(A_x, A_y), B(B_x, B_y), ...) tvořící ohraničující čtverec se stranou délky $2 * d$. Takto získané souřadnice použijeme pro vytvoření čtvercové obálky. Ve skutečnosti nám však stačí pouze dva protilehlé body (A, C), protože při vytvoření minimálního ohraničujícího obdélníku je zachována rovnoběžnost a kolmost stran.

Tím omezíme velký počet prohledávaných souřadnicových bodů jen na množinu těch, kteří spadají do této obálky. Toto je implementováno metodou `Queries.getPointsForMBR`, která vrací textovou WKT definici objekt `MultiPoint(MULTIPOINT(A, C))` pro zadanou středovou lokaci a vzdálenost d. Třída `Queries` je obslužná třída pro navazování a ukončování databázového spojení a pro vykonávání SQL dotazů a příkazů.

Souřadnice vyhovující kritériu maximální vzdálenosti vyhledáme pomocí `MBRContains(oblast, bod)`. Upozornění: MySQL 8.0 stále obsahuje chybu [59], která vyřadí použití indexu, pokud je výsledek funkce porovnán s nějakou boolean hodnotou (1, 0, TRUE, FALSE). V podmínce dotazu tedy musí být uvedeno pouze `MBRContains(oblast, bod)`, mohou však následovat další podmínky připojené pomocí AND a OR operátorů. Stejně se chovají funkce `MBRWithin`, `ST_Contains` a `ST_Within`.

Ve čtvercové obálce se však mohou nacházet body, jejichž vzdálenost od středu je větší než d. Dá se říct, že jde o výskyty spadající do rohů oblasti. Tyto musíme z výsledku odfiltrovat. Proto z užšího výběru ze čtvercové obálky vybíráme ty souřadnice, u kterých je vzdálenost menší nebo rovna d, a to pomocí volání funkce `ST_Distance_Sphere(střed, bod)`. Voláme ji pro každý bod v obálce. Toto je implementováno v metodě `Queries.getNearbyDealersByLocation`.

Parametry WKT funkce MULTIPOINT ve skutečnosti budou GPS souřadnice A, C čtverce. Hodnota `vfr_loc.g` představuje sloupec typu geometry se souřadnicemi bodů. Predikát je zobrazen na zkráceném výpise 6.2 a také v příkladech v příloze D na stránce 98.

```
WHERE MBRContains(ST_GeomFromText('MULTIPOINT(:body)', 4326), vfr_loc.g)
  AND ST_Distance_Sphere(ST_GeomFromText(:bod, 4326), vfr_loc.g) <= :d
```

Listing 6.2: Ukázka zápisu predikátu dotazu s využitím ohraničujícího obdélníku

Srovnání výkonů vyhledávání s a bez použití obálky

Toto srovnání již bylo uvedeno v kapitole 3.8.7 na stránce 40.

Možnost budoucí implementace pomocí ST_Buffer

Jakmile bude v MySQL realizovaná podpora geografických systémů i pro funkci `ST_Buffer`, bude možné ji využít pro vytvoření přesnější kruhové obálku pouze dle výchozího bodu (střed kruhu) a dle hledané vzdálenosti (poloměr kruhu). V predikátu pak předáme `ST_Buffer` jako parametr funkci `ST_Contains`, viz 6.3, kde další parametr bude opět sloupec s prohledávanými souřadnicemi. Odpadne tedy nutnost tvořit pomocný ohraničující polygon a porovnávat vzdálenost bodů v něm obsaženém, protože v případě kruhové plochy tvořené funkcí `ST_Buffer` je zaručeno, že všechny body ležící v kruhu tvoří množinu validních výsledků. I bez této nutnosti budeme stejně volat `ST_Distance_Sphere`, ne však v predikátu, ale pro prezentaci výsledků, kde chceme zobrazit i vzdálenost nalezených lokací od výchozího bodu.

```
WHERE ST_Contains(ST_Buffer(ST_GeomFromText(:bod, 4326), :d), vfr_loc.g)
```

Listing 6.3: Návrh predikátu při použití `ST_Buffer`.

6.3.2 Upřesnění výsledků dle délky silniční trasy

Výsledky vyhledávání dle vzdušné vzdálenosti je ještě nutné upřesnit, protože uživatel vyhledává dle maximální silniční vzdálenosti. Ta může být oproti vzdušné vzdálenosti až několikanásobně větší. Pro toto ověření a omezení sady mezivýsledků se v demonstrační aplikaci používá aplikace OSRM.

Volání OSRM funkce Route

Zpracování výsledků MySQL dotazu pro vyhledání lokací dle vzdušné vzdáleností je implementováno ve skriptu `_dealers_analysis_result.php`. Vrácená kolekce prodejců se v cyklu prochází a pro každý prvek se volá funkce Route z OSRM rozhraní. Pro podporu těchto volání je v demonstrační aplikaci připravena třída `OSRMRequest`, která pro funkci Route očekává v konstruktoru `withFromToLocations` parametry počátku a cíle trasy, tedy výchozího bodu a polohy každého z procházených bodů z předešlého výsledku.

Komunikace s OSRM API je založena na http protokolu a API se společně s předávanými vstupními parametry volá pomocí URL adresy, viz 6.4. Tato adresa je připravena v metodě `OSRMRequest.get Route`. Pro OSRM funkci Route je nastavení parametrů jednoduché – pouze se do URL přidají počáteční a cílové souřadnice oddělené středníkem. Pro optimalizaci je vhodné ještě potlačit vrácení

dodatečných informací, které v této chvíli nepotřebujeme, jako například množinu průjezdních bodů pro vizualizaci trasy, a to pomocí parametru `overview=false`.

```
http://router.project-osrm.org/route/v1/driving
    /18.11639,49.83712;18.09193,49.86628
```

Listing 6.4: Příklad volání funkce Route z OSRM API.

PHP funkce `file_get_contents` odešle připravenou URL a zároveň načte výsledek, který je vrácený jako odpověď na HTML požadavek. Odpověď z OSRM je ve formátu JSON, který se v servisní třídě `OSRMResult` převede na atributy objektu odpovídajícím požadovaným parametrům (doba trasy, délka trasy).

Pokud délka trasy nesplňuje prvotní limit zadaný uživatelem, je tento aktuálně procházený objekt vyjmut z finální kolekce výsledků.

6.3.3 Zobrazení výsledků na vložené mapě

Výsledky vyhledávání pomocí geolokačních dat jsou zobrazeny na stránce v tabulce seřazené dle vzdušné vzdálenost. Sloupce tabulky obsahují tyto informace: název prodejce, adresu prodejce, délku silniční trasy a dobu trasy.

Pro vizualizaci výsledků jsou nalezení prodejci zobrazeni také na vložené mapě.

Aktualizace mapy

Výsledky vyhledávání se předají ve formátu JSON do JavaScript funkce `showResultsOnMap`. Ta pro každý výsledek volá `addCoordOnMap` pro přidání značky představující nalezenou lokaci na vložené Leaflet mapě. Značka bude mít popisek s názvem prodejce, jeho poštovní adresu a délky a čas trasy z výchozího místa.

6.3.4 Zobrazení návrhu efektivní okružní trasy

Na stránce s finálními výsledky vyhledávání umožníme uživateli vyvolat akci pro návrh efektivní trasy pro navštívení všech nalezených lokací. Opět použijeme aplikaci OSRM.

Volání OSRM funkce Trip

Ještě před volání OSRM funkce Trip je nutné připravit si její vstupní parametry, a to na stránce zobrazující hlavní výsledky vyhledávání. Ze souřadnic výsledků se vytvoří kolekce průjezdních bodů `$tripWaypoints`. V kolekci bude zároveň uživatelsky zadaný výchozí bod, protože i on musí být součástí trasy pro navštívení nalezených míst. Kolekce se předává odesláním formuláře do PHP skriptu `_dealers_analysis_trip.php`.

Pro podporu volání OSRM je opět použita implementována třída `OSRMRequest`, která pro funkci Trip očekává v konstruktoru `withLocations` všechny průjezdní body hledané trasy. Konstruktoru se tedy předá kolekce `$tripWaypoints`.

URL adresa OSRM požadavku, viz 6.5, je připravena v metodě `OSRMRequest.getTrip`. Pro OSRM funkci Trip budou hlavním parametrem souřadnice průjezdních bodů oddělených středníkem. Očekáváme, že ve výsledku budou tyto průjezdní body seřazeny dle navrhovaného pořadí navštívení. Dále nastavíme parametry `overview=simplified` a `geometries=geojson`. U funkce Route jsme tyto informace ve výsledku potlačili, nyní je však využijeme. Dle těchto parametrů bude výsledná odpověď obsahovat i křivku ve formátu GeoJSON představující navrženou trasu. Křivku můžeme zobrazit na vložené mapě, která je s formátem GeoJSON kompatibilní.

```
http://router.project-osrm.org/trip/v1/driving
    /18.11639,49.83712;18.09193,49.86628?overview=simplified&geometries=geojson
```

Listing 6.5: Příklad volání funkce Trip z OSRM API.

PHP funkce `file_get_contents` odešle připravenou URL a zároveň načte výsledek, který je vrácený jako odpověď na HTML požadavek. Odpověď z OSRM je ve formátu GeoJSON, který uložíme do pomocného objektu `OSRMResult`. Pro zobrazení výsledků okružní trasy pak ve skriptu `_dealers_analysis_trip.php` umístíme křivky z GeoJSON objektu na vloženou mapu pomocí JavaScript funkce `showTripOnMap`. Ta provede deserializaci JSON textu na objekty představující výsledek OSRM volání, mezi kterými je i kolekce `trips`, jejíž první prvek obsahuje prostorový objekt geometry obsahující křivky tvořící nalezenou trasu. Pomocí volání Leaflet API funkcí `L.geoJSON(geojsonFeature).addTo(mymap)` přidáme prostorový objekt (vizualizaci trasy) na vloženou mapu, viz obrázek 6.3.

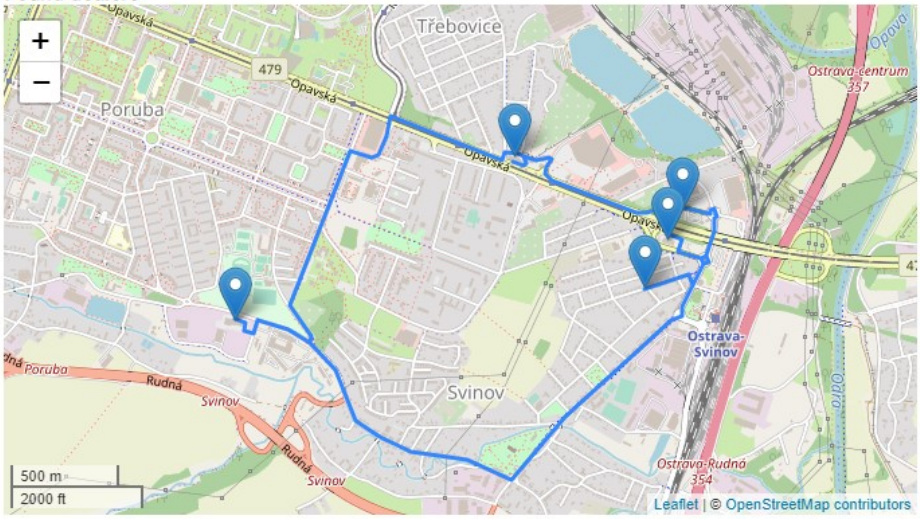
Dále z výsledku volání funkce Trip získáme výsledné pořadí zadaných průjezdních bodů a seřazené je zobrazíme v přehledové tabulce.

6.4 Možnosti rozšíření

Ve světle zavedených restrikcí v reakci na pandemii COVID-19 platných v čase přípravy tohoto textu (jaro 2021), se nabízí možnost rozšíření aplikace, využívající hranic okresů, které nelze bez patřičných důvodu opustit.

Hranice okrasu jsou definované jako prostorový GML objekt ve zdrojovém registru RÚIAN. V současné chvíli aplikace tyto informace neeviduje. Po úpravě datového modelu a aplikace importující číselníková data by bylo možné dále rozšířit parametry pro vyhledávání lokací prodejců tak, aby se vyhledání omezilo jen na příslušný okres výchozí polohy. Okres by představoval prostorový objekt Polygon a podmínky dotazu by obsahovaly navíc volání MySQL funkce `ST_Contains`, kde prvním parametrem by byl právě prostorový objekt představující hranici okresu. Touto prohledá-

Found dealers



Showing the optimal trip on the map

	Distance (km)	Name	Address	Route distance (km)	Route duration	Distance difference	Links
1	0.24	ADOP-car a.s.	Ostrava - Svinov, Opavská 348/11, 72100	0.69	01:51	0.45	OpenMaps Mapy.cz OSRM API
2	0.39	AUTOBOND GROUP a.s.	Ostrava - Svinov, Opavská 798/4, 72100	0.82	01:44	0.43	OpenMaps Mapy.cz OSRM API
3	0.85	AUTOCENTRÁLA s.r.o.	Ostrava - Třebovice, Třebovická 5534/1a, 72200	1.83	03:58	0.98	OpenMaps Mapy.cz OSRM API
4	1.89	KODECAR, spol. s r.o.	Ostrava - Poruba, Nad Porubkou 2355, 70800	3.14	05:28	1.25	OpenMaps Mapy.cz OSRM API

Obrázek 6.3: Ukázka nalezené okružní trasy

vanou (nad)oblastí by tedy bylo zaručeno, že se do výsledku nezahrne žádný prodejce spočívající v jiném okresu.

Další výzvou by však zůstalo upravení hledání silničních vzdáleností a efektivních okružních tras, které by nesměly ve svých návrzích a výsledcích protnout hranice okresu.

Kapitola 7

Závěr

V první části této práce byly shrnuty teoretické aspekty řešeného problému. Pro lepší pochopení principů pro práci s prostorovými daty v IT řešeních byla představena problematika reprezentace údajů o zeměpisných polohách a souřadnicových systémech. Dále byl popsán Registr územní identifikace, adres a nemovitostí (RÚIAN), jeden ze zdrojů geolokačních dat. Následně se teoretická část dokumentu soustředila na popis prostorových dat v IT řešeních s přihlédnutím ke specifikaci Simple Features Access organizace Open Geospatial Consortium, dle které je implementována funkcionalita pro práci s prostorovými daty v textu představených databázových systémů. Tyto systémy byly prezentovány právě z pohledu podpory prostorových objektů a funkcí s nimi pracujícími. Kapitola věnující se některým dostupným aplikacím třetích stran, které naleznou uplatnění v návrhu řešení této práce, uzavřela teoretickou část dokumentu. Byly zmíněné populární online mapové aplikace včetně knihovny Leaflet, sloužící k zobrazení vložené mapy na stránku webové aplikace. Nakonec byla představena knihovna OSRM pro hledání trasových informací, tedy údajů opírající se o znalosti o podobě silniční sítě.

Druhá část této práce se již věnovala praktickým aspektům navrhovaných řešení. Jako ověření konceptu byla vyvinuta a představena jednoduchá demonstrační aplikace, ve které je možno provádět požadované vyhledávání entit, které obsahují informace o svých geografických polohách. V popisu implementace byl nejdříve představen koncept navrhovaného řešení. Tento návrh byl rozdělen do tří oblastí řešení: příprava a využití číselníku dat obsahující zeměpisné souřadnice, vývoj funkcí umožňujících zadání výchozí lokace pro parametrizaci vyhledávání a konečně implementace samotného vyhledávání. To bylo nejdříve popsáno z hlediska databázové úrovně pro hledání dle vzdušné vzdálenosti a završeno upřesněním výsledků dle vzdáleností silničních tras pomocí aplikace OSRM. Text věnující se implementaci byl uzavřen popisem použití OSRM aplikace pro navržení efektivní okružní trasy mezi nalezenými lokacemi.

V sekci příloh tohoto dokumentu je k nalezení, kromě jiných informací, také příručka ve formě krokového návodu objasňující instalaci, přípravu a ovládání demonstrační aplikace.

Použitelnost a vhodnost navrhovaného řešení zadání práce byly ověřeny formou demonstrační

aplikace, která poskytuje efektivní vyhledávání geolokačních údajů nad rozsáhlými kolekcemi dat. Vyhledávácí funkce aplikace je snadno rozšiřitelná pro použití v jiných podobných informačních systémech. Například v databázovém dotazu lze rozšířit predikát o další podmínky vyhledávání, zahrnující i jiné než geolokační atributy. Své uplatnění na poli e-commerce portálů evidujících polohová data můžou najít i představené aplikace pro zobrazení mapových dat a hlavně trasovací nástroj, který využívá zdarma dostupná data Open Street Map a který může být provozován lokálně, bez nutnosti volat vzdálené aplikační rozhraní.

Literatura

1. *Coordinate systems* [IBM Documentation] [online]. 2014-10-24 [cit. 2021-03-31]. Dostupné z: https://www.ibm.com/docs/en/db2/11.1?topic=systems-coordinate.
2. *A guide to coordinate systems in Great Britain* [online]. Ordnance Survey, 2015-03 [cit. 2021-03-31]. Dostupné z: https://web.archive.org/web/20150924061607/http://www.ordnancesurvey.co.uk/docs/support/guide-coordinate-systems-great-britain.pdf.
3. *WGS 84 / Pseudo-Mercator - Spherical Mercator, Google Maps, OpenStreetMap, Bing, ArcGIS, ESRI - EPSG:3857* [EPSG.io: Coordinate Systems Worldwide] [online] [cit. 2021-04-23]. Dostupné z: https://epsg.io/3857.
4. *Pokročilé vyhledávání a GPS souřadnice* [Seznam Nápověda] [online] [cit. 2021-04-23]. Dostupné z: https://napoveda.seznam.cz/cz/mapy/vyhledavani/pokrocile-vyhledavani-gps-souradnice/.
5. STREBE, Daniel R. *Mercator projection Square* [online]. 2011-12-16 [cit. 2021-04-25]. Dostupné z: https://commons.wikimedia.org/wiki/File:Mercator_projection_Square.JPG.
6. *Souřadnicové referenční systémy* [ČÚZK: Geoportál] [online]. 2018-01-05 [cit. 2021-04-23]. Dostupné z: https://geoportal.cuzk.cz/(S(r1yf5efnyhf4ridwgzg4beho))/Default.aspx?mode=TextMeta&text=INSPIRE_ref_systemy&side=INSPIRE_dSady&head_tab=sekce-04-gp&menu=411.
7. *Souřadnicové systémy* [ČÚZK: Geoportál] [online]. 2018-05-05 [cit. 2021-04-23]. Dostupné z: https://geoportal.cuzk.cz/(S(hmen0s2gqkzq0foadvsckucj))/Default.aspx?mode=TextMeta&side=sit.trans&text=souradsystemy.
8. *WGS84 - World Geodetic System 1984, used in GPS - EPSG:4326* [EPSG.io: Coordinate Systems Worldwide] [online] [cit. 2021-04-23]. Dostupné z: http://epsg.io/4326.
9. *Mapy API verze 4.13 – Neil Armstrong* [API Mapy.cz] [online] [cit. 2021-04-23]. Dostupné z: https://api.mapy.cz/#pact.
10. *Databáze adres v ČR a číselníky územních celků* [Ministerstvo vnitra České republiky] [online] [cit. 2021-04-23]. Dostupné z: https://www.mvcr.cz/clanek/databaze-adres-v-cr-a-ciselniky-uzemnich-celku.aspx.

11. *Zákon o základních registrech*. 2009-03-26. Č. 111/2009 Sb. Dostupné také z: http://aplikace.mvcr.cz/sbirka-zakonu/ViewFile.aspx?type=c&id=5470.

12. *Adresy* [ČÚZK: Geoportál] [online]. 2021-04-23 [cit. 2021-04-23]. Dostupné z: https://geoportal.cuzk.cz/(S(scaux4gpgjcr4gzvzdsqt5sg))/Default.aspx?lng=CZ&mode=TextMeta&side=INSPIRE_dSady&menu=415&head_tab=sekce-04-gp&metadataID=CZ-00025712-CUZK_SERIES-MD_AD&metadataXSL=INSPIRE_Adresy.

13. FORMÁNEK, Jiří. Struktura a popis výměnného formátu RÚIAN (VFR). 2021-01-03, s. 7/32. Dostupné také z: https://www.cuzk.cz/Uvod/Produkty-a-sluzby/RUIAN/2-Poskytovani-udaju-RUIAN-ISUI-VDP/Vymenny-format-RUIAN/Vymenny-format-RUIAN-(VFR)/Struktura-a-popis-VFR-1_8_0.aspx.

14. *INSPIRE služba transformace souřadnic - úvod* [ČÚZK: Geoportál] [online]. 2020-01-18 [cit. 2021-04-23]. Dostupné z: https://geoportal.cuzk.cz/(S(40lnouquokatvix0plqetaeo))/Default.aspx?mode=TextMeta&text=sit.trans.uvod&side=sit.trans&head_tab=sekce-03-gp&menu=34.

15. KRÁTKÝ, Michal. *Databázové a informační systémy 2 - 11. Prostorová data a databázové systémy* [online]. 2020 [cit. 2021-04-23]. Dostupné z: https://dbedu.cs.vsb.cz/files/2020-2021/DAIS2/dais2-11a.pdf.

16. GAURA, Jan; ĎURÁKOVÁ, Daniela. *Reprezentace prostorových dat, prostorové modely (Geografické informační systémy)* [online]. 2018-04-12 [cit. 2021-04-23]. Dostupné z: http://wiki.cs.vsb.cz/images/9/9f/Gis_lecture3_11.pdf.

17. MASCIOTO, Matias. *Spatial Data with Python — Let's Begin!* [Medium] [online]. 2020-01-02 [cit. 2021-04-25]. Dostupné z: https://blog.rmotr.com/spatial-data-with-python-lets-begin-e29b5c41ead3.

18. *What is raster data?* [ArcMap | Documentation] [online] [cit. 2021-04-23]. Dostupné z: https://desktop.arcgis.com/en/arcmap/latest/manage-data/raster-and-images/what-is-raster-data.htm.

19. *OGC Members* [Open Geospatial Consortium Inc.] [online] [cit. 2021-04-23]. Dostupné z: https://www.ogc.org/ogc/members.

20. HERRING, John R. *OpenGIS® Implementation Standard for Geographic information - Simple feature access - Part 1: Common architecture*. Open Geospatial Consortium Inc., 2011-05-28. Dostupné také z: https://www.ogc.org/standards/sfa.

21. HERRING, John R. *OpenGIS® Implementation Standard for Geographic information - Simple feature access - Part 2: SQL option*. Open Geospatial Consortium Inc., 2010-08-04. Dostupné také z: https://www.ogc.org/standards/sfs.

22. STOLZE, Knut. SQL/MM Spatial: The Standard to Manage Spatial Data in Relational Database Systems. [N.d.], s. 18. Dostupné také z: http://doesen0.informatik.uni-leipzig.de/proceedings/paper/68.pdf.

23. *Chapter 5. PostGIS Reference* [online] [cit. 2021-04-23]. Dostupné z: https://postgis.net/docs/reference.html.

24. ROCCHINI. *Minimum bounding rectangle* [online]. 2012-05-23 [cit. 2021-04-25]. Dostupné z: https://commons.wikimedia.org/wiki/File:Minimum_bounding_rectangle.svg.

25. CALDWELL, Douglas R. *Unlocking the Mysteries of the Bounding Box* [online]. 2005-08-29 [cit. 2021-04-23]. Dostupné z: http://archive.ph/v03B.

26. GILLIES, Sean; BUTLER, H.; DALY, M.; DOYLE, A.; SCHAUB, T. *The GeoJSON Format* [online]. 2016-08 [cit. 2021-04-23]. Dostupné z: https://tools.ietf.org/html/rfc7946.

27. Geohash. In: *Wikipedia* [online]. 2021-02-22 [cit. 2021-04-23]. Dostupné z: https://en.wikipedia.org/w/index.php?title=Geohash&oldid=1008179817.

28. PORTELE, Clemens. *OGC® Geography Markup Language (GML) — Extended schemas and encoding rules* [online]. Open Geospatial Consortium Inc., 2012-02-07 [cit. 2021-04-23]. Dostupné z: https://www.ogc.org/standards/gml.

29. GUTTMAN, Antomn. R-Trees - A Dynamic Index Structure for Spatial Searching. 1984, s. 11. Dostupné také z: http://www-db.deis.unibo.it/courses/SI-LS/papers/Gut84.pdf.

30. KONINK, Stefan de; BAČA, Radim. *R-Tree example* [online]. 2010-04-06 [cit. 2021-04-25]. Dostupné z: https://commons.wikimedia.org/wiki/File:R-tree.svg.

31. *R-Tree* [online]. Simon Fraser University, [b.r.] [cit. 2021-04-23]. Dostupné z: https://www2.cs.sfu.ca/CourseCentral/454/jpei/slides/R-Tree.pdf.

32. RIGAUX, Philippe; SCHOLL, Michel; VOISARD, Agnès. *Spatial Databases: With Application to GIS*. 1st edition. San Francisco: Morgan Kaufmann, 2001-06-01. ISBN 978-1-55860-588-6.

33. BENTLEY, Jon Louis. Multidimensional binary search trees used for associative searching. *Communications of the ACM* [online]. 1975-09-01, roč. 18, č. 9, s. 509–517 [cit. 2021-04-23]. ISSN 0001-0782. Dostupné z DOI: 10.1145/361002.361007.

34. ROBINSON, John T. The K-D-B-tree: a search structure for large multidimensional dynamic indexes. In: *Proceedings of the 1981 ACM SIGMOD international conference on Management of data* [online]. New York, NY, USA: Association for Computing Machinery, 1981-04-29, s. 10–18 [cit. 2021-04-23]. SIGMOD '81. ISBN 978-0-89791-040-8. Dostupné z DOI: 10.1145/582318.582321.

35. PROCOPIUC, Octavian; AGARWAL, Pankaj K.; ARGE, Lars; VITTER, Jeffrey Scott. *Bkd-Tree: A Dynamic Scalable kd-Tree* [online]. 2003 [cit. 2021-04-23]. Dostupné z: https://users.cs.duke.edu/~pankaj/publications/papers/bkd-sstd.pdf.

36. ANDZIC, Mladen; STEIN, Steve; RAY, Mike; YOSHIOKA, Hiroshi. *Spatial Indexes Overview* [SQL Server | Microsoft Docs] [online]. 2016-12-09 [cit. 2021-04-23]. Dostupné z: https://docs.microsoft.com/en-us/sql/relational-databases/spatial/spatial-indexes-overview.

37. *Component overview* [OpenStreetMap Wiki] [online]. 2018-11-03 [cit. 2021-04-23]. Dostupné z: https://wiki.openstreetmap.org/wiki/Component_overview.

38. *11.4 Spatial Data Types* [MySQL :: MySQL 8.0 Reference Manual ::] [online] [cit. 2021-04-23]. Dostupné z: https://dev.mysql.com/doc/refman/8.0/en/spatial-types.html.

39. *MySQL/MariaDB Spatial Support Matrix* [MariaDB KnowledgeBase] [online] [cit. 2021-04-23]. Dostupné z: https://mariadb.com/kb/en/mysqlmariadb-spatial-support-matrix/.

40. *Chapter 4. PostGIS Usage* [online] [cit. 2021-04-23]. Dostupné z: https://postgis.net/docs/postgis_usage.html.

41. *Chapter 4. When to use Geography Data type over Geometry data type* [online] [cit. 2021-04-23]. Dostupné z: https://postgis.net/docs/manual-3.1/using_postgis_dbmanagement.html#PostGIS_GeographyVSGeometry.

42. NGUYEN, Thanh. Indexing PostGIS databases and spatial Query performance evaluations. *International Journal of Geoinformatics*. 2009-09-01, roč. 5, s. 1–9. Dostupné také z: https://www.researchgate.net/profile/Thanh-Nguyen-267/publication/294782438_Indexing_PostGIS_databases_and_spatial_Query_performance_evaluations/links/5cedd67c458515026a63860b/Indexing-PostGIS-databases-and-spatial-Query-performance-evaluations.pdf.

43. *The MongoDB 4.4 Manual* [online]. 2020-07-30 [cit. 2021-04-23]. Dostupné z: https://docs.mongodb.com/manual.

44. *Elasticsearch Guide [7.12] | Elastic* [online] [cit. 2021-04-24]. Dostupné z: https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html.

45. *Multi-dimensional points, coming in Apache Lucene 6.0* [Elastic Blog] [online]. 2016-02-15 [cit. 2021-04-23]. Dostupné z: https://www.elastic.co/blog/lucene-points-6.0.

46. MURRAY, Chuck; RAVADA, Siva; WANG, Jack; ANDERSON, Richard; HU, Ying; XIE, Qingyun (Jeffrey); HORHAMMER, Mike; ABUGOV, Dan; ALEXANDER, Nicole; BLACKWELL, Bruce; CHATTERJEE, Raja; COVARRUBIAS, Luis Angel Ramos; GERINGER, Dan; KAZAR, Baris; KOTHURI, Ravi; YANG, Ji; JAYAPALAN, Lavanya. *Spatial Concepts* [Oracle Help Center] [online]. 2021-03 [cit. 2021-04-25]. Dostupné z: https://docs.oracle.com/en/database/oracle/oracle-database/21/spatl/spatial-concepts.html#GUID-CD6ABC34-1CB4-476D-ACB1-BDA07026C206.

47. *Top U.S. mapping apps by users 2018* [Statista] [online]. 2021-02-04 [cit. 2021-04-23]. Dostupné z: https://www.statista.com/statistics/865413/most-popular-us-mapping-apps-ranked-by-audience/.

48. *OpenStreetMap Wiki* [online] [cit. 2021-04-23]. Dostupné z: https://wiki.openstreetmap.org/wiki/Main_Page.

49. *Plugins - Leaflet - a JavaScript library for interactive maps* [online] [cit. 2021-04-23]. Dostupné z: https://leafletjs.com/plugins.html.

50. *Using WMS and TMS services* [Leaflet - a JavaScript library for interactive maps] [online] [cit. 2021-04-23]. Dostupné z: https://leafletjs.com/examples/wms/wms.html.

51. *OSRM API Documentation* [online] [cit. 2021-04-23]. Dostupné z: http://project-osrm.org/docs/v5.23.0/api/#.

52. DIJKSTRA, E. W. A note on two problems in connexion with graphs. *Numerische Mathematik* [online]. 1959-12-01, vol. 1, no. 1, s. 269–271 [cit. 2021-04-24]. ISSN 0945-3245. Dostupné z DOI: 10.1007/BF01386390.

53. DANIEL J. ROSENKRANTZ; STEARNS, Richard E.; LEWIS Philip M., II. An Analysis of Several Heuristics for the Traveling Salesman Problem. *SIAM Journal on Computing* [online]. 1977-09-01, roč. 6, č. 3, s. 563–581 [cit. 2021-04-23]. ISSN 0097-5397. Dostupné z DOI: 10.1137/0206041.

54. ROSENKRANTZ, Daniel J.; STEARNS, Richard E.; LEWIS Philip M., II. *Traveling Salesman Problem - Farthest Insertion* [online]. 1974 [cit. 2021-04-23]. Dostupné z: https://www2.isye.gatech.edu/~mgoetsch/cali/VEHICLE/TSP/TSP015__.HTM.

55. ANTOŠ, David. *Co nám poví Našeptávač?* [Lupa.cz] [online]. 2006-06-20 [cit. 2021-04-27]. Dostupné z: https://www.lupa.cz/clanky/co-nam-povi-naseptavac/.

56. HINNER, Martin. *S-JTSK->WGS84* [Martin Hinner's homepage] [online] [cit. 2021-04-25]. Dostupné z: http://martin.hinner.info/geo/.

57. *Geolocation API* [online] [cit. 2021-04-24]. Dostupné z: https://w3c.github.io/geolocation-api/#introduction.

58. DREYER, Håkon. *Coordinate system - Algorithm for offsetting a latitude/longitude by some amount of meters* [Geographic Information Systems Stack Exchange] [online]. 2010-10-27 [cit. 2021-04-25]. Dostupné z: https://gis.stackexchange.com/questions/2951/algorithm-for-offsetting-a-latitude-longitude-by-some-amount-of-meters/2980#2980.

59. STAFFANS, Johannes. *MySQL Bugs: #76384: Spatial index not used when checking return values explicitly* [online]. 2015-03-19 [cit. 2021-04-24]. Dostupné z: https://bugs.mysql.com/bug.php?id=76384.

Příloha A

Zdrojové soubory implementovaného řešení

Součástí práce je i archív zdrojových souborů a skriptů. Jeho obsah je následující:

- `import_ciselniku`: obsahuje podadresáře se zdrojovým kódem a skripty pro vytvoření číselníku adresních míst a jejich GPS poloh.
  - `1_sql`: SQL skripty pro vytvoření schématu a pro dodatečné úpravy, které mají být spuštěny po provedení importu dat
  - `2_java`: projekt v jazyce JAVA představující aplikaci pro import číselníkových dat do databázového schématu aplikace
  - `3_vdp_data`: částečně připravené skripty pro stažení vstupních dat z RÚIAN
- `web_aplikace`: obsahuje podadresáře se zdrojovým kódem a skripty pro spuštění demonstrační webové PHP aplikace.

Příloha B

Příprava a instalace demonstrační aplikace

Tato kapitola představuje uživatelskou příručku popisující kroky pro prvotní spuštění demonstrační aplikace. Zdrojové soubory a skripty aplikace jsou součástí archívu, který je přiložen k této práci. Obsah adresářové struktury archívu byl představen v příloze A. K těmto adresářům se budeme v nadcházejícím popisu často vracet a budeme se odkazovat právě na cesty souborového systému, tak jak byly popsány.

Aby bylo implementované řešení připraveno k použití, je nutné vytvořit a naplnit lokální číselník adres, spustit na serveru webovou uživatelskou aplikaci včetně podpůrné aplikace OSRM.

B.1 Vytvoření číselníku

Nejdříve je nutné vytvořit aplikační číselník adres. Začneme vytvořením databázového schématu dle dodaného skriptu. Pak provedeme stažení zdrojových dat z VDP RÚIAN. Nakonec zkompilujeme JAVA aplikaci ImportRegistryData, která stažené XML data z registru analyzuje a relevantní obsah vkládá do databáze. Po dokončení tohoto importu je ještě nutné spustit skript pro dodatečné úpravy v databázi.

B.1.1 Ruční stažení aktuálních RÚIAN dat

Z VDP RÚIAN je nutné stáhnout dvojí typ dat. Proces se pro tyto případy liší jen nastavením vyhledávacího formuláře ve VDP (položka Územní prvky). První dávku pro vložení krajů a okresů (označenou jako Územní prvek: Stát až ZSJ) a druhou dávku pro vložení obcí (označenou jako Územní prvky: Obec a podřazené).

1. Použijeme formulář dostupný zde: https://vdp.cuzk.cz/vdp/ruian/vymennyformat/vyhledej
2. Nastavíme možnosti vyhledání takto:
    - Platnost údajů: Platné

- Časový rozsah: Úplná kopie
- *pro stažení krajů a okresů*
  - Územní prvky: Stát až SZJ
- *pro stažení obcí*
  - Územní prvky: Obec a podřazené
- Datová sada: Základní
- Výběr z údajů: Základní údaje
- Územní omezení: ČR

3. Zvolíme Vyhledat

4. Zvolíme Seznam linků. To vyvolá stahování textového souboru, který obsahuje seznam odkazů archívů ke stažení.

5. V případě obcí bude odkazů (řádků souboru) více než 18 tisíc. Nachází se zde totiž kromě posledních platných dat i dávky až 2 měsíce starší. Odkazy jsou setříděné od nejnovějších a datum dávky lze jasně odvodit z názvu odkazu, např. URL https://vdp.cuzk.cz/vymenny_format /soucasna/20210331_OB_500011_UZSZ.xml.zip odpovídá datu 31.3.2021. Odkazy starších dat ze souboru vymažeme, protože pro import číselíku nám stačí jen poslední platná verze. Tím se počet odkazů sníží na cca 6 tisíc.

6. Pomocí aktualizovaného souboru s odkazy stáhneme dávkově všechny archívy např. programem wget[1]:

   - `wget -i seznamlinku.txt -no-check-certificate -nc -c -directory-prefix =`*`cilovy_adr`*
   - řetězec *`cilovy_adr`* nahradíme nějakým jménem adresáře, kde se mají soubory ukládat

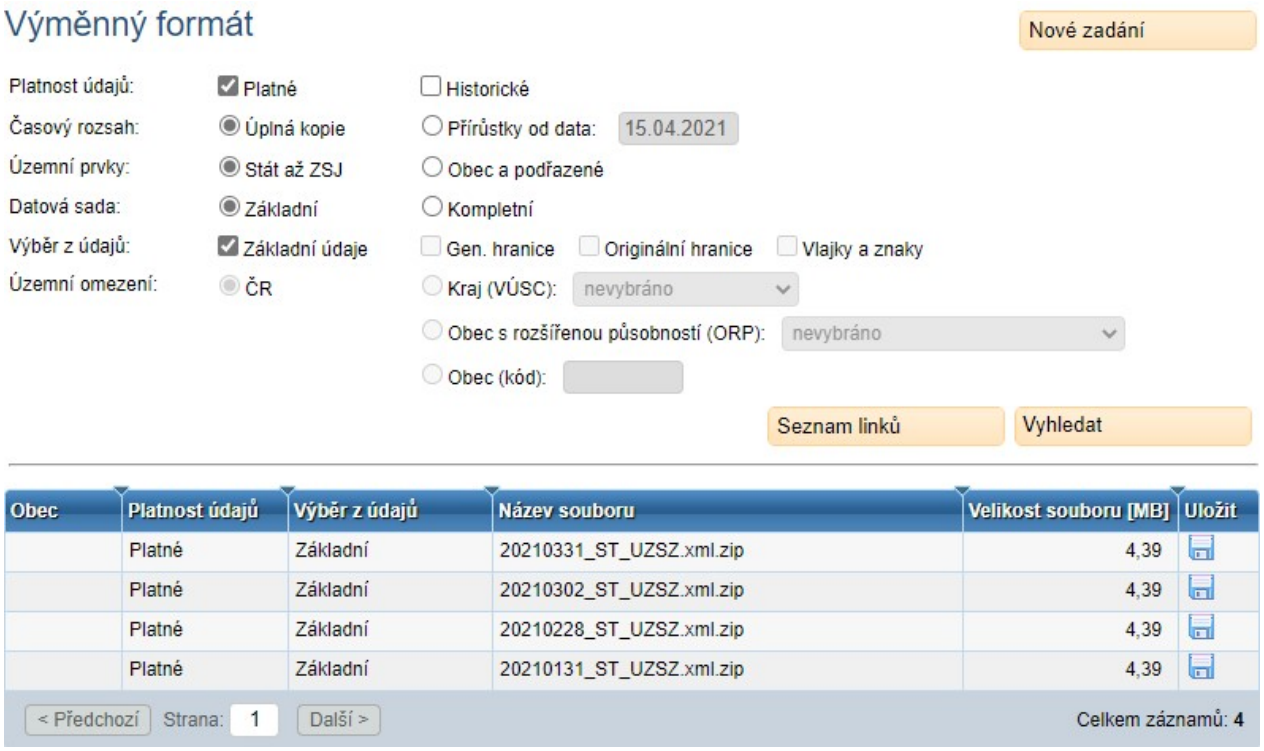
7. V archívu přiloženém k této práci se v adresáři `3_vdp_data` nachází skripty `run_wget_stat.bat` a `run_wget_obec.bat`, které lze použít pro stažení RÚIAN dat aktuálních ke dni 31.3.2021. Aby skripty fungovaly, musí být v globální PATH proměnné definována cesta k `wget.exe` souboru, nebo je možné ho umístit do adresáře se skripty.

8. Bude staženo více než 1,8 GB dat. I na rychlém připojení je celková rychlost limitována propustností zdrojového VDP RÚIAN serveru a může trvat i více než 50 minut.

9. Extrahujeme stažené archívy. Bude potřeba alespoň 44 GB diskového prostoru.

Tímto jsou data připravena pro import do DB.

[1] dostupný pro Windows zde: http://gnuwin32.sourceforge.net/packages/wget.htm



Obrázek B.1: Formulář na webu VDP

B.1.2 Příprava MySQL databáze pro import číselníkových dat

Navržené řešení vyžaduje použití MySQL databáze minimálně ve verzi 8.0. Zdrojové soubory se nachází v adresáři `1_sql`.

- V databázi vytvoříme schéma `dptest` pomocí skriptu `01 DDL - before import.sql`.
  - Výchozí název schématu (`dptest`) je možno změnit na prvních dvou řádcích tohoto skriptu. Pokud se tak stane, je nutné název aktualizovat i ve zbývajících SQL skriptech, které budou níže popsané. Název schématu je pak nutné změnit na prvních řádcích v příkazu `USE dptest;`.
  - Skript definuje pro tabulky úložiště InnoDB z důvodu objasněných v kapitole 3.8.1 o MySQL.
  - Sloupec `VRF_LOC.G` bude sloužit pro uložení souřadnic. Pro jeho definici je zadán i zvolený souřadný systém, a to následujícím způsobem: `G GEOMETRY NOT NULL SRID 4326`

B.1.3 Kompilace JAVA aplikace pro import číselníkových dat

Navržené řešení vyžaduje použití JAVA ve verzi 1.8 a předpokládá dostupnost aplikace maven. Zdrojové soubory se nachází v adresáři `2_java`.

1. Nejdříve upravíme zdrojový soubor `van179.orm.connConnectionPool.java`, kde na řádcích 17 až 23 zadáme do proměnných údaje pro navázání spojení s databází:
   - `schemaName` = název schématu (výchozí je `dptest`)
   - `initialConnString` = JDBC string pro navázání spojení s DB
   - `initialUser` = uživatelské jméno pro přihlášení do DB
   - `initialPwd` = heslo pro přihlášení do DB
2. Zkompilujeme pomocí maven aplikace příkazem: `mvn clean install`

B.1.4 Spuštění JAVA aplikace pro import číselníkových dat

Po kompilaci aplikace pro import číselníkových dat a po vytvoření DB schématu je možné přistoupit k posledním přípravám a k následnému spuštění aplikace.

1. V kořenovém adresáři aplikace vytvoříme adresáře `vdpxml/obec` a `vdpxml/stat`.
2. Do podadresáře `vdpxml/stat` vložíme soubor z VDP představující RÚIAN prvek Stát. Název souboru bude přibližně *YYYYMMDD*_ST_UZSZ.xml, kde prvních osm znaků představuje datum (rok, měsíc a den), ke kterému byla data z RÚIAN vygenerována. Soubor bude mít velikost přibližně 50 MB.
3. Do podadresáře `vdpxml/obec` extrahujeme všechny archívy představující prvky typu Obec. Formát názvů souborů je *YYYYMMDD*_OB_*123456*_UZSZ.xml, kde prvních osm znaků je opět datum a šestimístné číslo v názvu představuje identifikátor RÚIAN sady. Velikost všech extrahovaných souborů může být až 50 GB.
4. Aplikaci spustíme například pomocí aplikace maven příkazem:
   ```
   mvn exec:java
   -Dexec.mainClass=„van179.registryimport.ImportRegistryData"
   -Dexec.cleanupDaemonThreads=false
   ```

B.1.5 Úprava databáze po importu dat

Dále je nutné spustit několik SQL skriptů pro některé DML a DDL úpravy, které je vhodné provádět až po dokončení hromadného importu. Existence indexů a cizích klíčů by vkládání záznamů zpomalovalo, proto se tyto struktury a omezení vytvoří až po vložení dat. Dále se provedou některé dodatečné úpravy přímo v datech. Některé záznamy jsou totiž importovány duplicitně, jelikož jsou duplikované už ve vstupních souborech. Kontrola unikátnosti záznamů se při importu z důvodu rychlosti neprovádí. Je totiž daleko snadnější některé duplicity odhalit a smazat dodatečnými DELETE příkazy než unikátnost dat ověřovat při každé aktualizaci tabulky během importu.

Zdrojové soubory se nachází v adresáři `1_sql`.

1. Nad databází spustíme skript `02 DML.sql`. Skript smaže duplicitní záznamy v tabulkách číselníku.

2. Nad databází spustíme skript `03 DDL DML.sql`.

   - Skript vytvoří nad tabulkami indexy.
   - Ještě před vytvořením prostorového indexu nad tabulkou `VFR_LOC` dojde k úpravě dat v tabulkách `VFR_STAVEBNI_OBJEKT` a `VFR_LOC`. Smažou se záznamy, které jsou pro potřeby číselníku nadbytečné. Jde o stavební objekty, které ještě nemají přiřazené žádné adresní místo a nelze je tedy zahrnout do dotazu. Tyto záznamy nelze přeskočit při samotním importu číselníků, protože teprve po jeho kompletním skončení můžeme zjistit, zda se nějaké adresní místo na stavební objekt odkazuje.
   - Vytvoří se prostorový index nad tabulkou `VFR_LOC`. Tento příkaz je výkonnostně náročný a na vývojářské sestavě (viz poznámka 2 na stránce 40) trval 113 minut.

3. Nad databází spustíme skript `04 DML.sql`. Skript vytvoří nad tabulkami cizí klíče. Pro obsáhlou tabulku `VFR_ADRESNI_MISTO` může tato operace trvat až 40 minut.

B.1.6 Vložení testovacích dat

Pro demonstraci vyhledávání je nutné vytvořit záznamy představující prodejce. Spustíme skript, který vytvoří 4000 prodejců tak, že náhodně vybere adresní místa a k nim přiřadí záznam představující prodejce. Náhodnost výběru zajistí, že lokace vytvořených prodejců bude distribuována vzhledem k hustotě adresních míst, což odpovídá realitě, kdy i pobočky prodejců se koncentrují ve větších městech. Náhled na možný výsledek této operace je zobrazen na obrázku B.2.

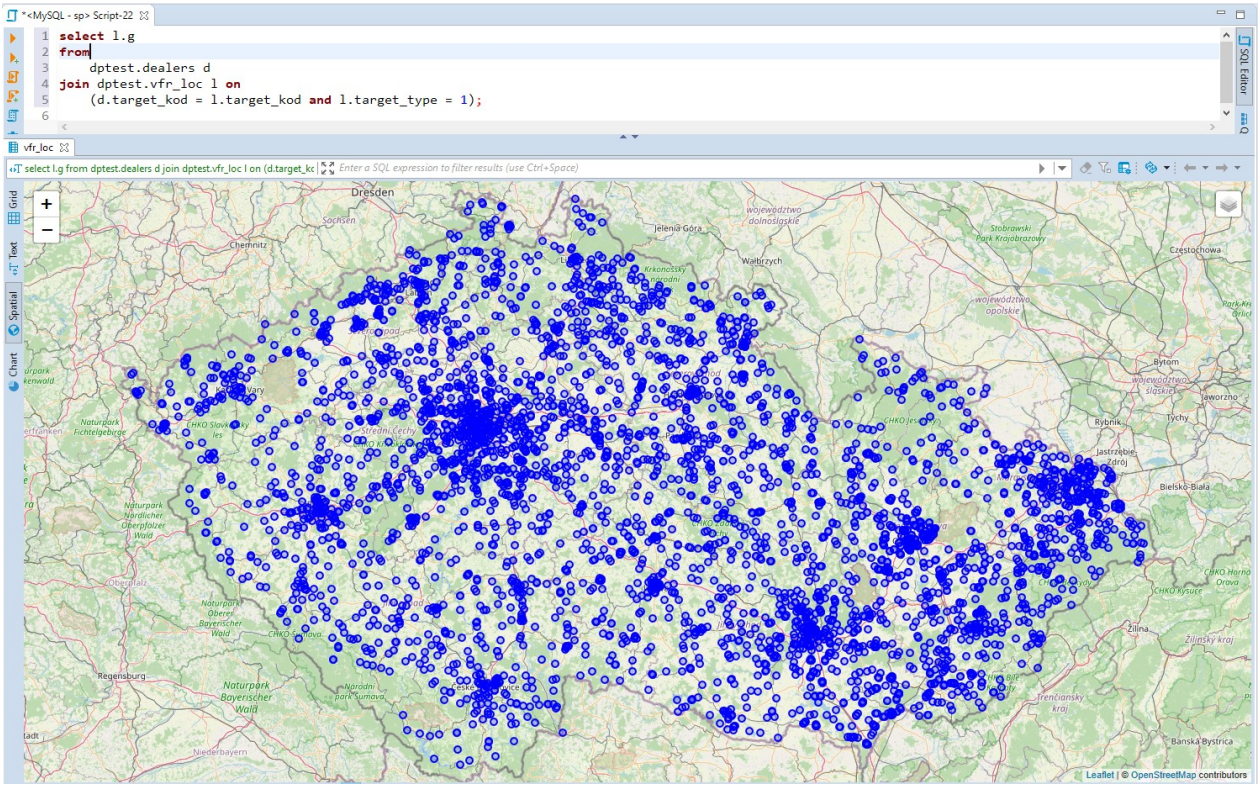
Zdrojové soubory se nachází v adresáři `1_sql`.

1. Nad databází spustíme skript `05 DML - create data.sql`.

   - Skript vytvoří 4000 náhodných prodejců, kterým bude přiřazeno adresní místo a pomocí dalších vazeb z tabulky `VFR_ADRESNI_MISTO` i kompletní adresa s GPS souřadnicemi.

B.2 Příprava a spuštění demonstrační webové aplikace

Po přípravě číselníku, který společně s tabulkou prodejců (DEALERS) představuje celé databázové schéma projektu, je možné pokračovat s nasazením demonstrační webové aplikace. Ta je napsána v PHP, je tedy nutné ji nasadit na adekvátní webový server. Proces nasazení aplikace zde nebude popisován. Budeme se věnovat hlavně konfiguraci aplikace.



Obrázek B.2: Ukázka výsledku náhodného vytvoření prodejců

B.2.1 Nastavení PHP aplikace

1. Hlavní úpravy konfigurace projektu provádíme v souboru `config.php` v adresáři `resources`. Konfigurace je tvořena asociativním polem, kde klíč určuje konfigurační element a jeho hodnota představuje zvolené nastavení. Nebudeme se v následujícím popisu orientovat dle čísel řádků, ale dle cesty tvořené klíči v tomto poli.

   (a) Upravíme údaje pro připojení k databázi. Měníme tyto hodnoty:

   - db.db1.dbname = název schématu číselníku, implicitně `dptest`
   - db.db1.username = uživatelské jméno pro přístup do databáze
   - db.db1.password = heslo pro přístup do databáze
   - db.db1.host = adresa („connection string“) databáze

   (b) Upravíme nastavení OSRM adresy:

   - urls.osrmUrl = jedná se o URL, na které bude dostupné rozhraní aplikace OSRM. Implicitně je nastaveno na http://localhost:5000.

B.2.2 Nasazení PHP aplikace

Aplikaci nasadíme na vhodný PHP server. Adresář obsahující skript `index.php` je `public_html`. Další kroky závisí na dostupném serveru a nebudeme se jimi blíže zabývat. Zmiňme pouze, že pro základní vývojový PHP server, který dostupný ke stažení z https://www.php.net, je příkaz pro nasazení aplikace: `php -S localhost:8000 -t public_html`.

B.2.3 Instalace OSRM aplikace a virtualizačního nástroje Docker

Aplikace Open Source Routing Machine je distribuována buď ve formě knihoven pro jazyk C/C++, nebo jako samostatná webová aplikace dostupná ve formě kontejneru pro virtualizační nástroj Docker. V dalších krocích tedy začneme s instalaci programu Docker, budeme pokračovat stažením OSRM dat pro území ČR a v závěru popíšeme způsob nasazení kontejneru v aplikaci Docker včetně jejího základního ovládání.

1. Z https://www.docker.com stáhneme Docker Desktop aplikaci a nainstalujeme ji. Po úspěšné instalaci budeme mít k dispozici v globální PATH proměnné příkaz docker.exe a také aplikaci Docker Desktop, grafické rozhraní pro správu docker kontejnerů.

2. Stáhneme obraz OSRM aplikace přímo pomocí funkce a příkazu docker (spuštěném v příkazové řádce): `docker pull osrm/osrm-backend`

   - tento příkaz uloží obraz aplikace do repozitáře docker, což můžeme ověřit příkazem: `docker images`

3. Stáhneme data extrahovaná z Open Street Map dostupná na stránce https://download.geofabrik.de. Konkrétně stáhneme archív pro Českou republiku ve formátu osm.pbf: https://download.geofabrik.de/europe/czech-republic-latest.osm.pbf.

4. Data uložíme do adresáře dle našeho uvážení. V tomto příkladě předpokládejme uložení dat do adresáře, jehož celá cesta je `C:\dev\docker\osrm\data`. Z následujících příkazů bude patrné, jak je způsob zápisu cesty transformován dle syntaxe aplikace docker.

5. Provedeme předzpracování dat pro OSRM aplikaci spuštěním tří příkazů:

   (a) `docker run -t -v c/dev/docker/osrm/data:/data osrm/osrm-backend osrm-extract -p /opt/car.lua /data/czech-republic-latest.osm.pbf`

   (b) `docker run -t -v c/dev/docker/osrm/data:/data osrm/osrm-backend osrm-partition /data/czech-republic-latest.osrm`

   (c) `docker run -t -v c/dev/docker/osrm/data:/data osrm/osrm-backend osrm-customize /data/czech-republic-latest.osrm`

- přepínač `-v` provede namapování data adresáře tak, aby byl dostupný přímo z virtualizovaného prostředí kontejneru po jeho spuštění. Je zadána reálná cesta (`c/dev/docker /osrm/data`), následuje dvojtečka a označení, pod jakým adresářem má být tato cesta dostupná uvnitř virtualizované aplikace (`/data`). V poslední části příkazu lze vidět, jak je odkazováno na konkrétní soubory, které se v tomto virtuálním adresáři nachází (`/data/czech-republic-latest.osm.pbf`)

6. Nyní už je OSRM aplikace připravena pro použití. Spustí se pomocí příkazu `docker run -t -i -p 5000:5000 -v /c/dev/docker/osrm/data:/data osrm/osrm-backend osrm-routed -algorithm mld /data/czech-republic-latest.osrm`

   - API je přístupné ve výchozím nastavení na adrese localhost:5050 a je ho možné odzkoušet přímo voláním některé z OSRM funkcí, např.:
     - http://localhost:5000/route/v1/driving/18.11613,49.83702;17.30033,49.35064
     - okno prohlížeče by mělo zobrazit neformátovaný JSON výpis počínající atributem a hodnotou „code:Ok“.

Tímto byly popsány všechny konfigurační a instalační kroky nutné pro spuštění demonstrační aplikace.

Příloha C

Ovládání demonstrační aplikace

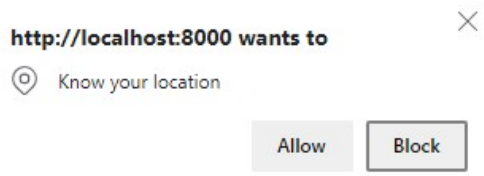
Po provedení všech kroků popsaných v předchozí kapitole, to jest po vytvoření číselníku, nastartování PHP serveru s nasazenou demonstrační aplikací a spuštěným docker kontejnerem s aplikací OSRM, je možno začít program používat.

1. K aplikaci přistoupíme pomocí webového prohlížeče na adresu zvolenou při nasazení na webovém serveru (např. http://localhost:8000).

2. V hlavním menu v záhlaví vybereme odkaz *Search*.

3. Nyní musíme vyhledávání parametrizovat vybráním výchozí polohy a maximální vzdálenosti pro nalezení prodejců. Máme následující možnosti:

   - Kliknutím na tlačítko *Get location* s nadpisem *Use current user's location* vybereme aktuální polohu, jak ji zprostředkovává zařízení, na kterém pracujeme. Po stisku tlačítka je nutné povolit prohlížeči požadavek pro získávání polohy, viz obrázek C.1.



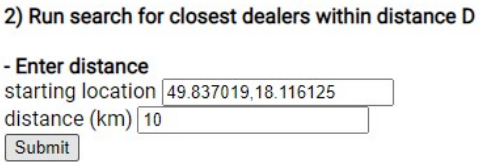
Obrázek C.1: Dotaz prohlížeče pro povolení získávání polohy

   - Další možností je zadání poštovní adresy pomocí našeptávače. Začneme psát adresu ve formátu „obec ulice číslo popisné“. Stačí zadat libovolný počet počátečních znaků těchto prvků oddělených mezerami. Místo názvu ulice jde vložit název části obce pro adresní místa bez přidělené ulice. Příkladem validního vstupu je „os poh 3“, který vrátí adresy s domovním číslem začínajícím číslem 3 v ostravských ulicích Pohoří (městská část

Krásné Pole) a Pohraniční (městské části Vítkovice, Moravská Ostrava), viz obrázek C.3. Nabízenou adresu vybereme pomocí ukazatele myši.

- Další možností je vybrat polohu přímo kliknutím do vložené mapy, viz obrázek C.4.
- Poslední možností je zadat ručně GPS souřadnice v pořadí *zeměpisná šířka a zeměpisná délka*, obě oddělené čárkou, a s tečkou, jako desetinným symbolem. Po zadání klikneme na tlačítko *Copy to form* pro vložení souřadnic do vstupního formuláře a pro aktualizaci vložené mapy, která bude na tyto souřadnice vystředěna.

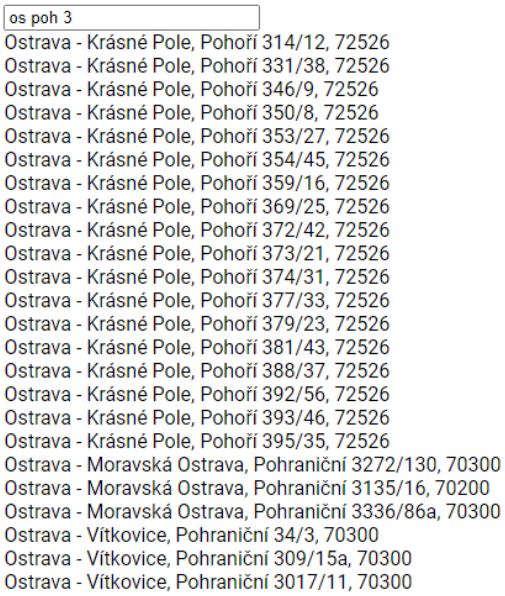
4. Po zvolení výchozí polohy ještě zadáme maximální hledanou vzdálenost v kilometrech, viz obrázek C.2.



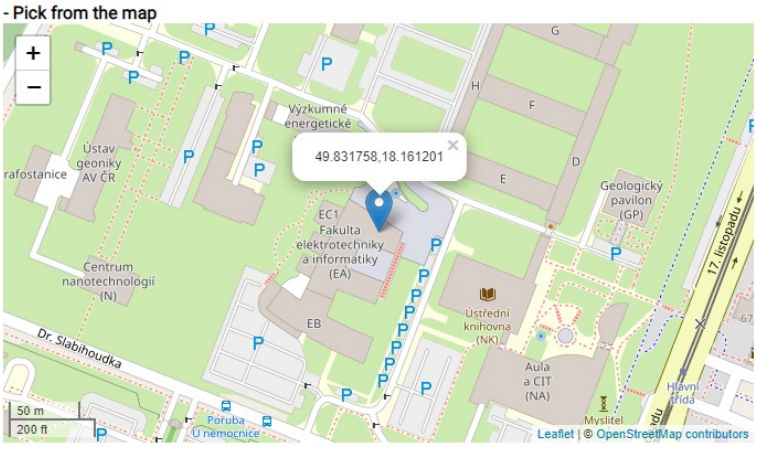
Obrázek C.2: Formulář vyplněný pro vyhledávání

5. Stisknutím tlačítka *Submit* vyvoláme vyhledávání. Výsledky budou zobrazeny na nově načtené stránce, viz obrázek C.5.

6. Pro nalezení efektivní okružní trasy klikneme na tlačítko *Show recommended trip*. Opět bude načtena nová stránka s výsledkem: na mapě uvidíme průjezdní body spojené křivkou značící trasu, viz obrázek C.6. Body budou seřazeny v přehledové tabulce dle pořadí, v jakém je efektivní je navštívit. Poslední bod bude zadaná výchozí poloha.

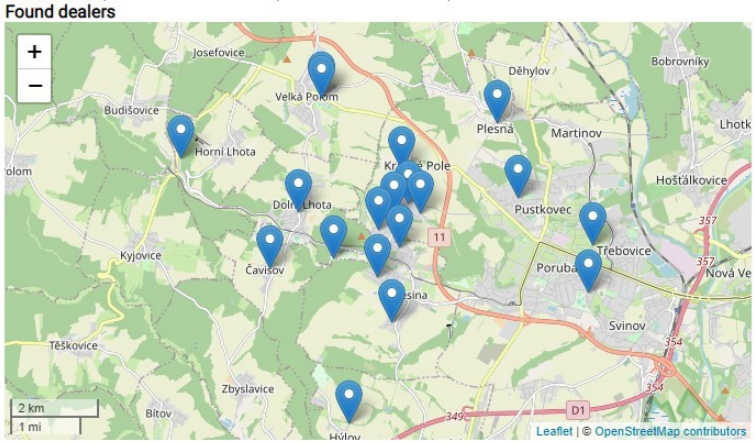


Obrázek C.3: Použití formulářového našeptávače



Obrázek C.4: Zobrazení výchozí lokace na mapovém prvku Leaflet

Found dealers

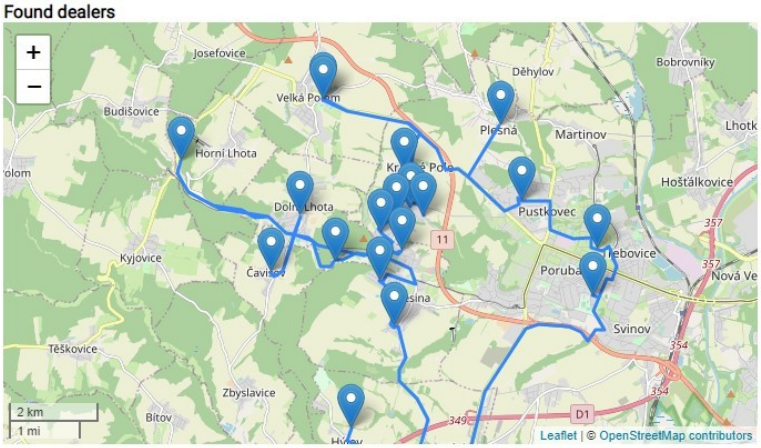


show recommended trip

	Distance (km)	Name	Address	Route distance (km)	Route duration	Distance difference	Links
1	0.49	rand3126455	Ostrava - Krásné Pole, Družební 225/121, 72526	0.64	01:38	0.14	OpenMaps Mapy.cz OSRM API
2	0.62	rand8370630	Vřesina - Vřesina, Slezská 606, 74285	0.83	02:04	0.22	OpenMaps Mapy.cz OSRM API
3	0.9	rand3128172	Ostrava - Krásné Pole, Nová kolonie 399/46, 72526	1.25	02:53	0.34	OpenMaps Mapy.cz OSRM API
4	1.01	rand25616498	Ostrava - Krásné Pole, Branecká 2030, 72526	1.88	04:07	0.87	OpenMaps Mapy.cz OSRM API
5	1.05	rand8370320	Vřesina - Vřesina, Topolová 575, 74285	2.68	06:32	1.62	OpenMaps Mapy.cz OSRM API

Obrázek C.5: Stránka s výsledky vyhledávání

Found dealers



Showing the optimal trip on the map

	Distance (km)	Name	Address	Route distance (km)	Route duration	Distance difference	Links
1	0.62	rand8370630	Vřesina - Vřesina, Slezská 606, 74285	0.83	02:04	0.22	OpenMaps Mapy.cz OSRM API
2	4.72	rand4400984	Horní Lhota - Horní Lhota, 1073, 74764	6.4	10:22	1.68	OpenMaps Mapy.cz OSRM API
3	1.84	rand4628781	Dolní Lhota - Dolní Lhota, Hořínůška 254, 74766	3.36	05:41	1.52	OpenMaps Mapy.cz OSRM API
4	2.58	rand4330463	Čavisov - Čavisov, Chrudimská 49, 74764	4.19	07:07	1.6	OpenMaps Mapy.cz OSRM API
5	1.19	rand27545091	Ostrava - Krásné Pole, Kyjovická 1046, 72526	1.96	03:59	0.77	OpenMaps Mapy.cz OSRM API

Obrázek C.6: Stránka s návrhem okružní trasy

Příloha D

MySQL dotazy

Tyto dotazy byly použity pro sestavení tabulky srovnání výkonu v MySQL a PostGIS v kapitole 3.8.7 na stránce 40. Nejde přímo o ukázku dotazu použitého v demonstrační aplikaci, byť v implementovaném řešení je použití prostorových funkcí v predikátu založeno na stejném principu, jako v případě prvního dotazu s MBR objektem.

První varianta pracuje s ohraničujícím obdélníkem a proto je použit prostorový index nad sloupcem VFR_LOC.G.

```
select *
  from vfr_loc l
 where MBRContains(
       ST_GeomFromText('MULTIPOINT(49.747187 17.976843,
                                   49.926850 18.255406)', 4326),
       l.g)
   and ST_Distance_Sphere(
        ST_GeomFromText('POINT(49.836798 18.115986)', 4326),
        l.g) <= 10000;
```

Listing D.1: Dotaz s MBR

Druhá varianta pouze porovnává všechny prostorové body s výchozím bodem zadaným souřadnicemi.

```
select *
  from vfr_loc l
 where ST_Distance_Sphere(
        ST_GeomFromText('POINT(49.836798 18.115986)', 4326),
        l.g) <= 10000;
```

Listing D.2: Dotaz bez MBR

Plán vykonávání prvního dotazu potvrzuje použití prostorového indexu `idx_vfr_loc_g`.

```
|select_type|type |possible_keys|key          |key_len|rows|filtered|Extra      |
|-----------|-----|-------------|-------------|-------|----|--------|-----------|
|SIMPLE     |range|idx_vfr_loc_g|idx_vfr_loc_g|34     |   6|   100.0|Using where|
```